VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro správu projektů

Project Management System

2015

Tomáš Varčok

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Varčok**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Systém pro správu projektů
Project Management System

Zásady pro vypracování:

Cílem práce je vytvořit webovou aplikaci pro správu projektů založenou na standardu HTML5. Součástí práce bude přehled existujících řešení pro online správu projektů. Aplikace bude umožňovat správu klientů a k nim přidružených projektů. U každého projektu bude možné definovat úkoly, chyby a další náležitosti spojené s řízením projektů. Aplikace bude také umožňovat komunikaci s klienty v podobě issue tracking systému.

Hlavní body zadání:
1. Přehled a srovnání existujících řešení v této oblasti.
2. Přehled použitých knihoven a nástrojů (představení jazyka TypeScript).
3. Návrh a implementace aplikace pro správu projektů.
4. Shrnutí dosažených výsledků.

Seznam doporučené odborné literatury:

[1] STOREKHEYROLLAHI, Tugberk Ugurlu; Alexander Zeitler; Ali. Pro ASP.NET Web API: HTTP Web Services in ASP.NET. New York, NY: Apress, 2013. ISBN 978-143-0247-258.
[2] LERNER, Ari. Ng-book - The Complete Book on AngularJS. Fullstack io, 2013. ISBN 978-0991344604.
[3] FENTON, Steve. TypeScript For JavaScript Programmers. lulu.com, 2013. ISBN 1291107371.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2015 .....

Rád bych poděkoval panu Ing. Janu Janouškovi za odborné vedení, rady a konzultace při zpracování této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá vytvořením systému pro správu projektů, který slouží k efektivnímu a jednoduchému uchovávání souvisejících informací, jejich sdílení, atd. Systém je tvořen jako webová aplikace s využitím technologií TypeScript, ASP.NET, Entity Framework, HTML5, jQuery a dalších.

V první kapitole je představeno několik existujících systémů pro správu projektů. Dále jsou popsány technologie a nástroje, pomocí kterých byl systém vytvořen. V další kapitole je detailněji rozebrána implementace systému, jeho celkové rozdělení a jednotlivé komponenty. Většinou je daná část představena nejprve obecně a poté je podrobněji popsána její implementace.

Klíčová slova: správa projektů, TypeScript, informační systém, webová aplikace

Abstract

This bachelor's thesis deals with creating project management system, which is used to efficient and easy storing of related information, sharing them, etc. The system is made as a web application using technologies TypeScript, ASP.NET, Entity Framework, HTML5, jQuery and others.

The first chapter presents several existing systems for project management. Then there is a description of technologies and tools, which were used to create the system. In the next chapter there is detailed analysis of the implementation of the system, its parts and single components. Each part is usually introduced in general and then there is a description of its implementation.

Keywords: project management, TypeScript, information system, web application

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
XML	–	eXtensible Markup Language
JSON	–	JavaScript Object Notation
UX	–	User eXperience
AJAX	–	Asynchronous JavaScript and XML
ANSI	–	American National Standards Institute
SQL	–	Structured Query Language
ASP	–	Active Server Pages
URL	–	Uniform Resource Locator
ORM	–	Object-relational mapping

Obsah

1 Úvod

Ani řízení organizace malé rodinné oslavy nemusí být ničím jednoduchým. Je potřeba sepsat seznam potřebných věcí a úkolů, naplánovat různé činnosti, atd. Máme-li na tuto přípravu například 3 osoby, hned může jít vše snadněji. Nebo také ne! Pak totiž máme nejen množství úkolů, ale také lidí, aktivních činitelů, kteří mohou něco dělat. Mohou konat dobře, ale i špatně, pokud například nefunguje komunikace a 2 udělají totéž a jiný (důležitý) úkol zůstane nesplněn.

A nyní přejdeme k trochu obtížnějšímu prostředí, jakým je například firma o 70 zaměstnancích. Místo hostů na oslavě zde figurují klienti, kteří neprominou téměř žádnou chybu. Namísto 3 lidí máme několik desítek zaměstnanců, z nichž drtivá většina nevyvíjí žádnou samostatnou aktivitu a pouze čekají na přidělení úkolů s přesnými instrukcemi. Jednotlivé úkoly již nemusí být vůbec jednoduché a mohou se skládat z dalších úkolů, mohou na sobě navzájem záviset, shlukovat se do souvisejících skupin, atd.

Právě v tomto prostředí je již nutné využívání technik takzvaného project managementu neboli projektového řízení (projektové správy). Tyto techniky často souvisí s komunikací, sdílením informací, ... To vše ale potřebuje i jistou oporu pro účely uchovávání informací a jejich sdílení, k čemuž sice lze využívat i pouhý telefon a papír, ale v dnešní době již tuto roli přebírají různé informační systémy.

Jde o systémy malého až obrovského rozsahu, které dokáží evidovat různé informace, přehledně rozdělovat práci, hlídat termíny, atd. K informacím mohou lidé přistupovat okamžitě, kdykoliv, současně, z různých míst a lze také aplikovat určité limity, kdo se k daným informacím dostane a kdo je může editovat. V datech lze různě vyhledávat a filtrovat je dle určitých vlastností. Obvykle lze v rámci takovýchto systémů také komunikovat. Toto poskytuje široké spektrum možností a při správném využívání dokáže takovýto systém značně pomoci firmě jako celku i každému jednotlivci, který jej bude využívat.

Tato práce se bude věnovat vytvoření systému, který spadá do popsané kategorie systémů pro správu projektů. Vytvořený systém by měl umožňovat evidenci všech informací souvisejících s evidencí a řízením projektů.

Nejprve zmíním některé existující systémy tohoto typu, v následující kapitole nastíním využité technologie a nakonec detailněji rozeberu návrh a implementaci vytvářeného systému.

2 Existující systémy pro správu projektů

V následující části budou popsány některé existující systémy pro správu projektů. U každého systému budou uvedeny obecné informace, čím je systém odlišný, co je v něm zajímavého, atd.

2.1 Basecamp

Basecamp [1] je webová služba pro správu projektů, která byla spuštěna již roku 2004 (nová, aktuální verze byla spuštěna 2012), takže jde o systém s delší historií.

Stojí za ním stejnojmenná mezinárodní společnost, která systém provozuje po celém světě. Navzdory tomu je Basecamp podle oficiálních stránek dostupný pouze v angličtině, ačkoliv texty by mělo být možné zadávat ve všech jazycích.

Již na domovské stránce jsou přehledně zobrazeny výhody systému. Velmi účinným lákadlem je takzvaná „Tour“, která ukazuje příběh, jak byl s pomocí Basecampu realizován větší projekt. Celý příběh je proložen mnoha obrázky a popisem, co bylo potřeba udělat a jak to vedoucí projektu udělali (přidání lidí, nahrání dokumentů, sdílení fotek z mobilu, atd.). Pro méně zkušené uživatele může být atraktivní výuka práce se systémem zdarma.

Registrace do systému je velice rychlá a jednoduchá. Nejsou vyžadovány žádné speciální údaje. Taktéž vnitřní uspořádání systému a design působí velkým důrazem na jednoduchost. První dojem také vylepší okénka s nápovědou a krátká videa vysvětlující službu celkově a následně konkrétní oblasti (základní fungování projektů, jak pozvat spolupracovníky či klienty, jak nahrávat soubory, apod.).

Přehled projektů může mít různou podobu podle volby uživatele - „dlaždice“, seznam dle abecedy, nebo kombinovaný přehled. U každého projektu lze evidovat diskuze, seznamy úkolů („to-do listy“), soubory, textové dokumenty a události. V celkovém detailu projektu jsou tyto jednotlivé entity seskupeny v blocích, ve kterých je vždy zobrazeno několik nejaktuálnějších položek a zbytek lze zobrazit po rozkliknutí.

Diskuze mají vzhled a funkčnost podobnou jako běžná fóra, přidání zprávy je jednoduché a je umožněno základní formátování jako tučné písmo, kurzíva, citace, seznam a číslovaný seznam. Taktéž je možné vkládat přílohy pomocí vybrání souboru z disku, přetažením metodou drag & drop, nebo zvolením souboru ze služby Google Drive (po nezbytném napojení Google účtu). To-do listy slouží ke stanovení a přiřazení úkolů a jsou, stejně jako celý systém, pojaty velmi jednoduše. U jednotlivých položek v listu lze přiřadit uživatele, stanovit termín splnění (deadline) a rovněž připojit přílohy. Soubory v projektu lze nahrávat již zmíněným způsobem a jsou zobrazeny v „dlaždicovém“ přehledu ve formě náhledů. Přehled obsahuje veškeré soubory nahrané v rámci projektu (tedy například v diskuzi nebo k položce to-do) a do místa jejich použití se lze jednoduše přemístit. Rovněž je možné jedním kliknutím k souborům založit diskuzi. Textové dokumenty jsou vytvářeny přímo v systému pomocí zjednodušeného textového vstupu. Pro rychlejší orientaci v systému zde hned po registraci existuje vzorový projekt s větším množstvím zanesených údajů a entit (jako To-do listy, soubory, atd.).

Velice dobrou funkcí je kalendář, ve kterém jsou automaticky zapsány všechny položky, které mají nějaké časové omezení (např. datum dokončení). Také je možné přidávat vlastní události. Kalendář může fungovat v módu zobrazení klasického kalendáře, nebo v módu „Agenda“, kde jsou události řazeny po dnech v přehledném seznamu. Praktickou funkcí je také možnost zobrazit si v kalendáři pouze události vybraného projektu.

Přehled provedených činností je možno nalézt v sekci „Progress“, kde jsou přehledně znázorněny aktivity, které uživatel provedl jako například splnění položky To-Do, přidání komentáře, nahrání souboru, atd. Celý přehled je příjemným způsobem rozčleněn dle dní i projektů.

Další uživatelsky přívětivou funkcí jsou šablony projektů, které umožní nadefinovat určitý výchozí projekt, ze kterého budou následně vycházet nové. Do šablony je možné nadefinovat vše, co do normálního projektu (To-Do listy, soubory, dokumenty, atd.), což následně usnadní vytváření podobných projektů v budoucnu.

Určitě je také vhodné zmínit nabízené API pro vytváření aplikací třetích stran, jako jsou například mobilní aplikace a desktop widgety.

Na tak dlouho působící aplikaci je poměrně zarážející podpora pouze jednoho jazyka (angličtiny), což pravděpodobně značně limituje větší rozšíření systému. Množství textů v systému není velké, takže by zde překlad minimálně do 3 dalších světových jazyků neměl chybět, i kdyby podpora byla poskytována pouze v angličtině. Systém také není zjevně stavěn na evidenci informací o klientech, což by v případě využití pro firmy mohlo vadit.

Pro lepší vyzkoušení a zvyknutí si na práci se systémem je určitě vhodná 2 měsíční zkušební doba, namísto standardního jednoho měsíce. Dalším zajímavým krokem je licence pro učitele zdarma pro účely výuky, která může Basecamp lépe zviditelnit. Kromě výjimky pro učitele Basecamp neposkytuje žádný bezplatný tarif. Ceny začínají na 20$ měsíčně za maximálně 10 aktivních projektů a 3 GB prostoru. Následně jsou dostupné dražší tarify, které se liší pouze vyšším limitem aktivních projektů a velikostí prostoru. Neomezený počet uživatelů je zahrnut ve všech tarifech.

2.2 Projektově.CZ

Projektově.CZ [2] je český nástroj pro řízení projektů. Za projektem stojí společnost Projektově.CZ s.r.o.

Po úvodním přihlášení nechybí představení systému formou okének, která popíší uživatelské rozhraní. Průvodce lze spustit také kdykoliv později v rámci aktuálně otevřené sekce. Design systému je velice jednoduchý.

Vypadá to, že se celý systém zaměřuje téměř výhradně na úkoly, čemuž je přizpůsobeno mnoho jeho částí (design, navigace, ...). Úkoly jsou zpracovány velmi dobře, obsahují dostatek nastavení a informací. Lze k nim přidávat komentáře i soubory. Rovněž je možné přidávat podřazené úkoly. Zajímavou možností je export úkolu do dokumentu formátu PDF.

Systém po prvotním otestování nepůsobí příliš jako „systém pro správu projektů“, ale spíše jako „systém pro správu úkolů“, protože soustředění na úkoly je velice znatelné. Přehled projektu sice poskytuje jisté možnosti, ale téměř všechny se týkají úkolů, které

do projektu patří. Projekt se tak nejeví jako podstatná entita v systému, která by měla nějaké vlastnosti, protože k němu lze zapsat v podstatě pouze název, popis, zkratku a dvě položky související s „cíli“. Soubory lze nahrávat výhradně jen k úkolům a ne obecně k projektu, což by se mohlo hodit například pro projektovou dokumentaci.

Projekt je tedy reálně pouze jakousi schránkou na úkoly. Jednak do něj lze úkoly přidávat a pak zde existují nástroje na úrovni projektu, které poskytují informace a souhrny o obsažených úkolech (jde tedy znovu o funkcionalitu související s úkoly). Ohledně projektů tedy systém nenabízí mnoho možností a soustředí se v podstatě pouze na úkoly, avšak v této oblasti je systém zpracovaný kvalitně a použitelnost je velmi dobrá.

Na některých místech je využit AJAX (ukládání, přiřazování úkolů, nastavení), avšak někde poněkud překvapivě ne. Když už je AJAX někde v systému použit, tak například při přepínání mezi typy úkolů a jejich řazení, kdy struktura (tabulka) zůstává stejná a mění se jen data, by použití AJAXu dávalo rovněž smysl.

Projektově.CZ zjevně nenabízí žádnou bezplatnou verzi (kromě zkušební na 30 dní). Cena základního tarifu je 990 Kč za měsíc pro 5 uživatelů. Vyšší tarif se liší počtem uživatelů a objemem uložených dat. Rovněž zde existuje individuální tarif na míru.

2.3 Bugzilla

Bugzilla [3] je systém, který najde uplatnění při procesu vývoje software. Oficiálně je prezentován jako „Defect Tracking System“ nebo „Bug-Tracking System“, tedy volně přeloženo jako systém pro sledování chyb. Umožňuje sledovat chyby a změny, komunikovat se spolupracovníky, zveřejňovat opravy (záplaty, „patche“), ... Celý systém je poskytován zdarma pod licencí *Creative Commons License*. Bugzilla je původně v anglickém jazyce, ale je dostupná i čeština. Pro označení hlášení chyb v rámci systému budeme dále používat také anglické zažité pojmenování „bug“.

Každá chyba (bug) nese mnoho informací, které jsou potřebné k jejímu nahlášení, správnému dohledání a vyřešení. Hlavním zacílením je produkt, komponenta, verze a případně URL, ke kterým se chyba vztahuje. Nechybí také informace o platformě uživatele (PC, mobil, ...) a verze operačního systému. Bugu lze nastavit určitou prioritu, stav, termín splnění (deadline) a klíčová slova pro vyhledávání. Taktéž lze přiřadit uživatele a jako zajímavá možnost se jeví nastavení závislosti na jiném bugu. K bugu lze přidávat zprávy v rámci komentářů.

Také je možné nahrávat přílohy. Jako určité možnosti specifické pro tento druh systému jsou zde k vidění například možnost označit přílohu jako záplatu („patch“) anebo označit nějakou starší přílohu jako zastaralou a nahrazenou tou aktuálně nahrávanou.

K přehledu práce slouží několik položek jako odhad pracnosti (celkový i aktuální), odpracované hodiny a další. Při vytváření nového bugu (zprávy o něm) je výhodné, že se během psaní názvu provede vyhledání již evidovaných položek, což uživatele nabádá k jednoduchému zjištění, zda nezakládá duplicitní zprávu. Systém umožňuje poměrně široké nastavení e-mailových notifikací, kde uživatel může přesně specifikovat, při jakých událostech má být informován.

V základní verzi vypadá systém celkově poměrně zastarale. Není ani patrný takový důraz na jednoduchost a uživatelskou přívětivost (UX). Jako výhoda může být vnímáno

to, že systém je určen pro vlastní nasazení, což znamená, že jej lze nainstalovat na místo podle vlastního uvážení a možností. Výhodou zde může být to, že v takovém případě není nutné spoléhat na poskytovatele služby a jeho síťovou infrastrukturu, a také z hlediska bezpečnosti může být tato varianta vnímána lépe (i když to nemusí být vždy oprávněné). Navíc je zde také možnost vlastních úprav uživatelského rozhraní, což je jistě vhodné, pokud nemá být systém používán pouze interně v rámci firmy.

2.4 Shrnutí

V této kapitole bylo představeno několik systémů patřících do kategorie systémů pro správu projektů. Existuje mezi nimi poměrně veliké množství rozdílů, přičemž jedním z nich je jisté odlišné zacílení. Basecamp působí jako univerzální systém, který zvládá poměrně vše a nabízí široké možnosti. Projektově.CZ je silně zaměřeno na práci s úkoly, což může v některých případech přesně odpovídat potřebám firmy. Bugzilla je zase striktně zaměřena na podporu procesu vývoje software. Basecamp a Projektově.CZ fungují na principu „Software as a Service“ (SaaS) a jsou tedy provozovány na serverech poskytovatele. Bugzilla je nástroj, který je nutné někde zprovoznit.

Všechny systémy mají podobné znaky a cíl (správa projektů), ale každý z nich má tedy do jisté míry odlišné zaměření a další vlastnosti, což ilustruje různorodost celé této kategorie informačních systémů.

3 Použité technologie a knihovny

V této kapitole budou představeny technologie použité k implementaci systému. Při vývoji bylo použito více různých jazyků, technologií, knihoven a dalších existujících komponent.

3.1 Typescript

Typescript [4] je poměrně nový programovací jazyk, který vznikl roku 2012. Jedná se o jakousi nadstavbu nad jazykem JavaScript. JavaSript je velice rozšířeným řešením pro tzv. client-side scripting, tedy provádění dynamických operací na straně uživatele (přímo v prohlížeči, nikoliv na serveru). JavaScript má ovšem jisté nevýhody, které práci v něm řadě programátorů znepříjemňují. I kvůli tomuto vznikl TypeScript.

Jak bylo řečeno, TypeScript je nadstavbou nad JavaScriptem. To prakticky funguje tak, že TypeScript kód je přeložen do JavaScript kódu. TypeScript kód tedy není spouštěn, pouze je v něm vytvářena požadovaná funkcionalita, přičemž lze využívat výhod, které TypeScript poskytuje, ale nakonec stejně dojde k překladu napsaného kódu do JavaScriptu. Díky tomuto principu se TypeScript řadí k takzvaným transpilerům.

3.1.1 Transpiling

Pro lepší pochopení pojmu *transpiling* bude vhodné se nejprve zastavit u pojmu kompilace. Kompilace znamená překlad, kdy je kód napsaný v jednom jazyce přetransformován do jiného jazyka. Transpiling je vlastně to samé, jen s tím rozdílem, že při něm jsou oba jazyky (zdrojový i výsledný) na podobné úrovni abstrakce.

Tento rozdíl si lze například ilustrovat na jazyku C# a CIL (Common Intermediate Language) kódu, do kterého je C# kód při kompilaci překládán. Psát kód v CIL by asi bylo možné, ale v dnešní době prakticky velmi nepohodlné a zdlouhavé, protože není pro psaní a čtení člověkem příliš vhodný. Příkladem transpilingu je právě TypeScript a JavaScript, kdy oba tyto jazyky jsou na podobné úrovni abstrakce a v JavaScriptu se tak běžně dají vytvářet i rozsáhlé aplikace.

Dalšími příklady transpilerů mohou být například Dart (do JavaScriptu), CoffeScript (do JavaScriptu), C++ na C, atd.

3.1.2 Výhody a novinky TypeScriptu

Z výše uvedeného principu fungování plyne, že TypeScript nemůže mít výrazný vliv na výkon při běhu programu (je-li zanedbán možný menší nárůst objemu kódu). Jeho síla je ale ve značném urychlení psaní kódu, kdy je programátorovi umožněno pružnější a méně náročné uvažování. Taktéž samotné psaní kódu je pohodlnější díky principům a postupům, na které jsou programátoři zvyklí z jiných programovacích jazyků.

Patrně by bylo vhodné poznamenat, že JavaScript opravdu usnadnění pro programátory potřebuje. O JavaScriptu někteří programátoři říkají, že je to takový „Assembler

pro web“. Toto prohlášení je samozřejmě nadsázkou, protože nerespektuje některé technické vlastnosti (např. závislost na platformě, snadná čitelnost JavaScriptu člověkem), ale příčinou vzniku této analogie je především ne zrovna snadný vývoj pro programátora. Je patrně možné to nazvat ne příliš dobrou „programátorskou přívětivostí“. Pravděpodobně neexistuje člověk, který měl s JavaScriptem co dočinění a moc dobře si nepamatoval mnohá úskalí, jakožto i pocity bezmoci až vzteku, když v něm něco vyvíjel a nebyl schopen najít chybu v komplikovanější struktuře kódu. Naštěstí dnes existuje mnoho JavaScript frameworků či transpilerů, které tuto práci usnadňují.

Jelikož je TypeScript nadstavbou nad JavaScriptem, tvoří standardní JavaScript jeho podmnožinu. Prakticky to znamená, že v TypeScript kódu je možné naprosto volně používat JavaScript. Toto umožňuje snadnou přenositelnost již napsaných scriptů a také to přináší pohodlí pro programátory (mohou stále používat části toho, na co byli zvyklí a co mají zažité).

Mezi nejvýraznější novinky, které TypeScript přináší, patří jistě podpora tříd a rozhraní, jak je vývojáři znají z jazyků jako je Java a C#. Toto je oproti JavaScriptu obrovská výhoda, která posunuje možnosti tvorby rozsáhlejších webově založených systémů na vyšší úroveň, protože programátorům poskytuje účinný strukturovací mechanismus, který lze jednoduše aplikovat. Díky podobnosti s dalšími objektově orientovanými jazyky je navíc usnadněn i návrh, protože vývojáři a návrháři jsou často zvyklí navrhovat funkcionality a celé systémy pomocí rozložení na jednotlivé třídy. Navíc je zde možnost využít takzvané moduly, které jsou podobné jmenným prostorům (namespace, jako v C#).

JavaScript je dynamicky typovaný jazyk, což znamená, že datové typy proměnných neurčuje vývojář, ale určují se automaticky podle obsahu, který je proměnné přiřazen. TypeScript přináší typový systém, kterým do zmíněného prostředí zavádí výhody staticky typovaných jazyků, což znamená, že vývojář zadává datové typy proměnných, parametrů i návratových hodnot funkcí ručně přímo do kódu. Jelikož je JavaScript podmnožinou TypeScriptu, statické typování je pouze volitelné a nikoliv povinné. Jinak by totiž neplatilo, že jakýkoliv JavaScript kód je zároveň TypeScript kódem (vývojář by musel všude ručně dopsat datové typy). Statické typování přináší přehlednější kód, ve kterém se může vývojář snadněji vyvarovat chyb souvisejících například s nesprávným předpokladem o datovém typu určité proměnné.

Hlavní výhody ovšem díky statickému typování mohou poskytovat vývojářské nástroje - IDE (Integrated Development Environment). Ty totiž poskytují nástroje urychlující vývoj a přinášející větší pohodlí pro vývojáře. Příkladem mohou být našeptávání a automatické doplňování při psaní kódu. Přejmenování proměnné či funkce je možné provést automaticky a bezpečně. Navigace v kódu je usnadněna příkazy jako *Go To Definition* a *Find All References*, které umožní se téměř okamžitě dostat do potřebné oblasti kódu bez nutnosti dlouhého uvažování, ve kterém souboru se nachází.

Zmíněné nástroje však naštěstí nejsou dostupné pouze pro vlastní TypeScript kód, ale je zde možnost je využívat i pro komponenty třetích stran, které mohou být napsány v čistém JavaScriptu. Toto je umožněno díky souborům obsahujícím deklaraci nabízeného API dané komponenty. Soubor s příponou „.d.ts“ obsahuje deklarace (nikoliv definice)

obohacené o statickou typovost parametrů a návratových hodnot funkcí, což je přesně to, co umožní IDE poskytnout výše uvedené výhody. K rozšířenějším komponentám již tyto deklarační soubory naštěstí existují. Obrovskou knihovnu lze nalézt na adrese [5].

3.1.3 Nevýhody TypeScriptu

Při přechodu z JavaScriptu na TypeScript jistě vývojáři spatří mnoho výhod, ale narazí i na jisté obtíže. Tou hlavní může být nutnost psát datové typy, abychom dosáhli na všechny výhody silné statické typovosti, což může být přinejmenším obtěžující. S některými komponentami třetích stran může nastat problém spojený právě s kontrolou typovosti. Deklaračních souborů (.d.ts) již existuje mnoho, ale některé nemusí být úplně dokonalé, nebo pro danou komponentu nemusí být žádný dostupný. Pak je nutné buď napsat vlastní, nebo se spokojit s řešením, které kontrolu typovosti pouze obejde za cenu ztráty výhod statické typovosti. Vše zmíněné však nemusí být problém, pokud se vývojář spokojí s omezením výhod, které mu TypeScript nabízí. To je sice rozporuplný postup, který však umožňuje vyhnout se těmto nevýhodám.

3.2 AJAX

Již od doby počátku používání webových stránek a aplikací u nich existuje poměrně nepříjemný problém, jehož závažnost narůstala postupně s růstem využívání grafiky, nových technologií a s tím spojeným růstem velikosti stránek. Tímto problémem je nutnost opětovného načtení celé stránky při nějakém požadavku, který vyžaduje interakci se serverem. Důsledkem toho je snížená interaktivita směrem k uživateli, který musí poměrně často zbytečně vyčkávat. Dalším problémem, který tento model přináší je to, že ze serveru je vždy zasílána znovu celá stránka, ačkoliv se změnila pouze její část. Například při změně řazení dat v tabulce je tak při obnovení stránky ze serveru znovu zasláno i záhlaví, menu, zápatí i další části obsahu, které se ale vůbec neliší od toho, co měl uživatel načteno před požadavkem na změnu řazení. Tím tedy narůstá objem přenášených dat a s tím spojená doba čekání uživatele. Uživatelé si již na tento model fungování a čekání pravděpodobně zvykli, což ale neznamená, že není potřeba se snažit o vylepšení.

Pro řešení tohoto problému bylo vyzkoušeno mnoho možností, z nichž většina ale vykazovala jisté vady (např. nutnost nainstalovat modul do prohlížeče). Jejich principem byla nějaká forma komunikace se serverem bez nutnosti aktualizovat stránku. Poté se objevila technologie AJAX (Asynchronous JavaScript and XML), jejíž počátky sahají již do roku 1999, kdy ale nebyla podpora ze strany prohlížečů tak široká. Výhody AJAXu oproti metodám, které se objevily dříve jsou poměrně zásadní - pracuje snad ve všech moderních prohlížečích, nevyžaduje instalaci žádného doplňku či modulu a dokáže spolupracovat s řadou technologií na straně serveru (PHP, ASP.NET, ...), což umožňuje zavést jeho používání i do již existujících komplexních aplikací. [6]

AJAX umožňuje poměrně jednoduchou asynchronní komunikaci se serverem na pozadí. Koncept požadavek-odpověď mezi klientem a serverem již s použitím AJAXu nevyžaduje obnovení stránky. Při jakékoliv akci uživatele je sestrojen objekt XMLHttpRequest,

který je jádrem AJAXu. Poté je na server odeslán požadavek spolu s volitelnými argumenty, které technologie na straně serveru může potřebovat k navrácení správných dat. Serverová technologie (např. ASP.NET) provede určité operace (které mohou zahrnovat přístup k uložišti, databázi, vzdálenému zdroji, ...) a poté odešle odpověď. Data odpovědi jsou typicky ve formátu XML nebo JSON. Poté, co jsou na klientské straně vytažena data z odpovědi, je možné je promítnout do stránky za pomocí dynamické modifikace modelu DOM.

Mezi typické příklady využití AJAXu patří například validace formulářových polí, asynchronní vyhledávání (či takzvané „našeptávání"), filtrace souhrnu dat dle nastavených parametrů, atd.

3.3 ASP.NET

ASP.NET je server-side technologie, jež funguje na straně serveru a umožňuje tvorbu dynamických webových stránek. De facto jde o framework, který umožňuje programátorovi využít jazyky C# nebo Visual Basic pro tvorbu webových aplikací, ačkoliv běžně jsou tyto jazyky určeny spíše pro desktopové aplikace. Výhodou ASP.NET je tedy to, že například i desktopový vývojář v C# dokáže přejít na tvorbu webových systémů.

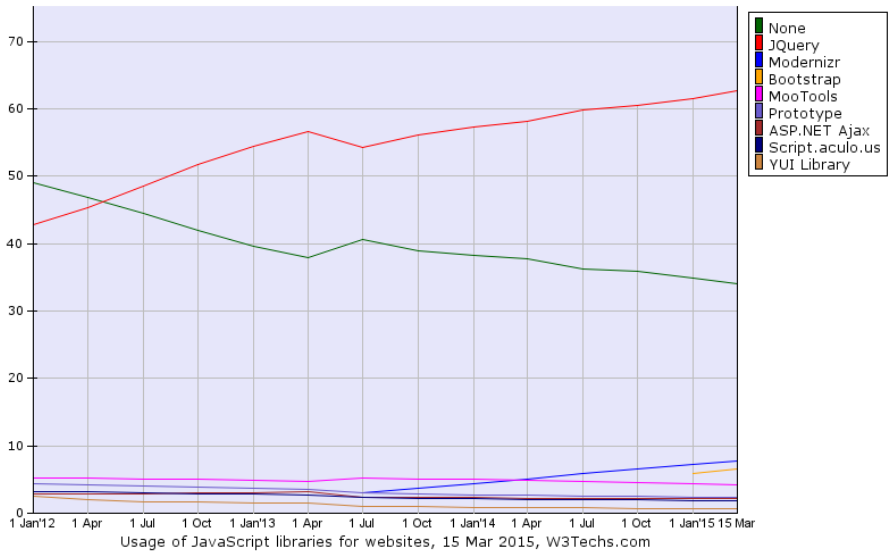
ASP.NET poskytuje řadu hotových řešení pro běžně implementované záležitosti jako je například autentifikace uživatelů, lokalizace, atd. Jelikož tato technologie běží na .NET frameworku, jsou taktéž dostupné jeho funkce jako například práce s XML, s databází, soubory, apod. Právě dostupnost všech rozšiřujících funkcí je velikou výhodou. U vývoje malých aplikací se sice tak moc neprojeví, ale u komplexnějších systémů ušetří spoustu času. Škála produktů využívajících ASP.NET je poměrně široká a může začínat u malých webových stránek a končit u obrovských korporátních systémů.

ASP.NET nabízí pro tvorbu webu 3 základní přístupy - Web Pages, WebForms a MVC. V tomto systému bylo využito WebForms. Toto řešení totiž plně dostačuje tomu, jak a k čemu je ASP.NET v systému použit, protože ve většině případů funguje jako prostředník mezi klientskou vrstvou a databází. K tomuto účelu je také využit Entity Framework, který je popsán dále. Rovněž je v systému využito mnoho možností .NET frameworku jako například autentifikace uživatele, práce s formátem JSON, práce se soubory, atd. [7]

3.4 Knihovna jQuery

Jak již bylo zmíněno v části věnované jazyku TypeScript, takzvaně „čistý" JavaScript není pro programátory zrovna jednoduchý. Jedním z možných pomocníků pro programování v JavaScriptu jsou knihovny.

Nejprve by bylo namístě popsat, co to vlastně knihovny jsou, přičemž bude také nastíněn vztah s transpilery (do JavaScriptu), které jsou popsány výše. Rozdíl mezi transpilery a knihovnami nemusí být každému na první pohled patrný a někdo může obojí považovat za „lepší a snadnější JavaScript". Knihovny se skládají z částí kódu, jež jsou sepsány v daném jazyce a které většinou poskytují programátorovi hotovou funkcionalitu (např. provádění animací, zprostředkování komunikace pomocí technologie AJAX) ve formě nějaké třídy, metody či funkce. Jemu pak už jen stačí vědět, co například dané funkci



Obrázek 1: Trendy podílů využívání JavaScriptových knihoven (s podílem větším než 1%) na webu [8]

předat a co lze očekávat za výstup, přičemž vnitřní implementací (jak je tato funkcionalita dosažena) se vůbec nemusí zabývat. Transpilery jsou oproti tomu novým jazykem s vlastními konstrukcemi a pravidly.

Mezi JavaScriptové knihovny patří například jQuery, Prototype, Dojo, ... Jak ilustruje obrázek 1, jQuery [9] je dlouhodobě naprostým favoritem mezi těmito knihovnami.

Knihovna jQuery, stejně jako její alternativy, se snaží především dosáhnout usnadnění práce programátora. Výslednou funkcionalitu vytvořeného kódu bychom dokázali dosáhnout i s programováním pouze v JavaScriptu, avšak díky použití této knihovny je to mnohem jednodušší, rychlejší a pohodlnější. Taktéž díky sníženému množství napsaného kódu a dobrému pojmenování funkcí je celková orientace jednodušší.

Knihovna jQuery dokáže usnadnit mnohé. Výhody je možné začít pociťovat již na nejnižší úrovni práce, jakou je přístup k modelu DOM, který knihovna zjednodušuje, zpřehledňuje a poskytuje vhodně pojmenované a dobře fungující metody pro manipulaci s DOM. Velice důležité pro interakci s uživatelem jsou události, jejichž zpracování jQuery rovněž zjednodušuje a rozšiřuje. Mnoho programátorů jistě uvítá zefektivnění práce s technologií AJAX, která se díky jQuery stala stručnější, přehlednější a tím pádem mnohem přitažlivější pro ty, kteří se dříve možná zalekli poměrně složitějšího zápisu práce s AJAXovými požadavky. Na jQuery lze rovněž přenést starosti ohledně kompatibility v různých prohlížečích, což je problematika, která v minulosti vedla k poměrně nenávistným vztahům webových vývojářů k jistým webovým prohlížečům.

V neposlední řadě je zde výhoda, která není přímo součástí jQuery jako takového - jde o značné množství zásuvných modulů. Mnoho jich lze najít například na adrese [10]. Tyto moduly může vytvářet v podstatě kdokoliv a lze je pak nabídnout k používání celému

světu. V případě, kdy programátor potřebuje docílit jisté funkcionality, je velice dobrý nápad zkusit nejprve vyhledat, zda pro tuto funkcionalitu již někdo nevytvořil modul, který by bylo možné využít.

3.5 Použité JavaScriptové moduly a komponenty

V této práci bylo pro dosažení určitých funkcionalit použito několik existujících JavaScriptových a jQuery komponent či modulů.

Pro zobrazení obsahu v takzvaném „lightboxu“, tedy způsobem, kdy je obsah zobrazen v jiné vrstvě stránky, zatímco originální obsah je touto vrstvou překryt, byl využit FancyBox [11]. Díky tomuto principu může uživatel provést nějaké akce v nové vrstvě a po jejím zavření je stále na původní stránce, kde jsou například zachovány všechny dříve vyplněné hodnoty.

Jelikož práce s formáty datumů je v JavaScriptu poměrně obtížná, pro převody mezi formáty byla využita komponenta Moment [12], která činí tuto práci velice přehlednou a snadnou.

Pro vykreslení grafů bylo použito Google Charts [13], což je služba, která umožňuje z vlastních dat vykreslovat různé typy grafů (sloupcové, koláčové, ...). Vše probíhá dynamicky za běhu pomocí JavaScriptu a nejde tedy o žádné předem vygenerované obrázky.

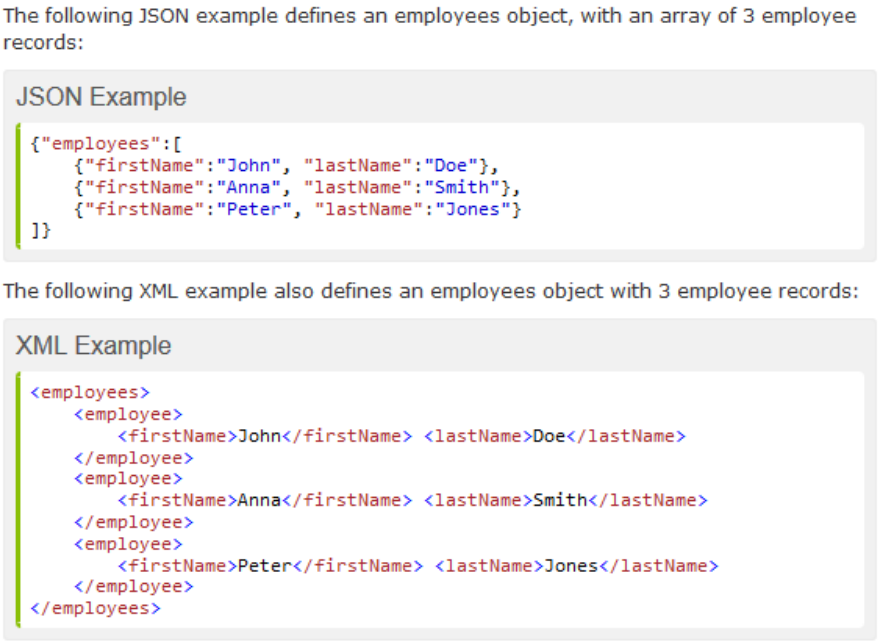
3.6 Formát pro přenos dat - JSON

V dnešním světě informačních technologií je potřeba často přenášet nebo uchovávat strukturovaná data. Dříve využívané textové formáty (txt, csv) nejsou již dnes z mnoha důvodů příliš vhodné, protože sice obsahují data, ale již neobsahují sémantiku (nebo ji obsahují ne úplně vhodným způsobem), tedy význam jednotlivých dat. Velkým přínosem byl příchod technologie XML (eXtensible Markup Language), která byla vyvinuta s cílem přinést softwarově i hardwarově nezávislý prostředek pro přenášení dat spolu s jejich sémantikou (popisem). XML je poměrně dobře strojově čitelné a výhodou je i poměrně snadná čitelnost pro člověka, která umožňuje využívat XML například pro konfigurační soubory.

Nevýhodou XML pro použití při přenosu dat je jeho velikost. Obsahuje totiž větší množství nadbytečného obsahu, který není nezbytně potřeba. Ačkoliv se XML stalo de facto standardem pro uchovávání dat v textové podobě a jejich přenášení, pro účely přenosu dat jej v současnosti již zřejmě porazil formát JSON. Ukázku srovnání formátů JSON a XML lze vidět na obrázku 2.

JSON (JavaScript Object Notation) je formát určený pro přenos textových dat. Z důvodu tohoto specifického určení je u něj kladen větší důraz na velikost, která je oproti XML zredukována díky syntaxi (nikoliv díky nějaké formě komprese). JSON využívá dvou struktur - kolekci párů název/hodnota (lze reprezentovat jako objekt, strukturu, slovník, atd.) a seznam hodnot (lze reprezentovat jako pole, seznam, atd.).

Kromě menší velikosti je výhodou JSON oproti XML také to, že je jednodušší na strojové zpracování, protože se lépe mapuje na datové struktury moderních programovacích



Obrázek 2: Ukázka srovnání formátů JSON a XML [14]

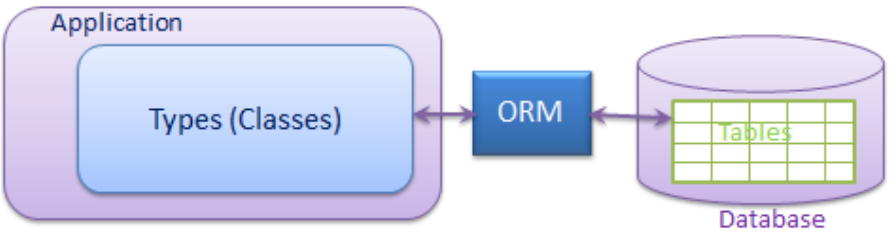
jazyků. Stejně jako XML jde o otevřený formát se silnou nezávislostí, protože je podporován v mnoha programovacích jazycích jako C#, Java, Python, PHP, samozřejmě JavaScript, atd. Tyto jazyky často poskytují programátorům nástroje pro relativně snadnou a efektivní práci s JSON.

Ve vytvořeném systému je JSON používán pro přenos dat mezi serverovou a klientskou vrstvou. Díky velmi dobré podpoře na straně jazyků TypeScript a C# je práce s JSON poměrně snadná.

3.7 ORM - Entity Framework

ORM neboli objektově-relační mapování je technika, která zajišťuje zpřístupnění relačních dat (uložených v relační databázi) do prostředí objektově-orientovaného programování. Možností, jak ORM realizovat je více, což lze ilustrovat například na existenci několika návrhových vzorů v knize Martina Fowlera [15], jakými jsou Table Data Gateway, Row Data Gateway, Active Record a Data Mapper. Velmi jednoduchá ilustrace ORM je znázorněna na obrázku 3.

Praktický přístup programátora k ORM může být dvojí - může ORM vytvořit sám (ideálně za použití nějakého nárhového vzoru), nebo může využít existující ORM nástroje. Oba tyto přístupy mají své klady i zápory, z nichž ty hlavní jsou uvedeny níže.



Obrázek 3: Jednoduchá ilustrace ORM [16]

- Vlastní realizace ORM
  - \+ Může být rychlejší za běhu (záleží na správném vytvoření)
  - \- Implementace ORM zabere nemalé množství času.
  - \- Případná změna v databázi nebo databázového systémů si vyžádá značný zásah do implementace ORM.
- Použití existujících ORM nástrojů
  - \+ Zavedení do systému, používání a změny jsou relativně snadné.
  - \+ Vývojář se namísto implementace ORM může věnovat aplikační business logice.
  - \+ Většinou lze snadněji změnit používaný databázový systém.
  - \+ Při změně v databázi (např. přidání atributu do tabulky) stačí aktualizovat mapování.
  - \- Může být pomalejší za běhu.

Při rozhodování mezi vlastní implementací ORM a existujícím ORM nástrojem je tedy nutné zvážit poměrně široké spektrum skutečností jako například požadavky na výkon vyvíjeného systému, počet uživatelů, jejich průměrnou zátěž na databázi, atd. V neposlední řadě bude jistě ve firemním prostředí rozhodovat i cena, tedy zda se firmě vyplatí pověřit programátora vývojem vlastního ORM vzhledem k rozpočtu projektu.

V implementovaném systému není očekávána veliká náročnost na výkon, vysoký počet uživatelů a není zde ani předpoklad manipulace s velkými objemy dat. Z těchto důvodů je využit existující ORM nástroj, kterým je Entity Framework, což je open source framework vyvinutý společností Microsoft. Podle převzaté a volně přeložené definice, původně dostupné v anglickém jazyce na stránce [16] lze Entity Framework popsat následovně:

Definice 3.1 *Microsoft ADO.NET Entity Framework je framework pro Objektově/Relační Mapování (ORM), který umožňuje vývojářům pracovat s relačními daty jako s doménově-specifickými objekty. Také odstraňuje většinu potřeby kódu pro přístup k datům, který jsou vývojáři obvykle nuceni napsat. Použitím Entity Framework mohou vývojáři pokládat dotazy za použití LINQ, načež*

následně obrdží silně typované objekty, se kterými mohou dále pracovat. ORM implementace Entity Frameworku poskytuje mnoho služeb, takže se vývojáři mohou více soustředit na aplikačně-specifickou business logiku, než na základy přístupu k datům.

Uvedená definice je velice výstižná. Velký důraz je v ní kladen na hledisko usnadnění práce vývojářů, což je nesporná výhoda. Nesmíme ovšem zapomínat na nevýhody oproti vlastní implementaci, které jsou rozebrány výše.

3.8 Bootstrap

Bootstrap [17] (dříve Twitter Bootstrap) je front-end framework, což znamená, že jde o jakousi sadu nástrojů a šablon, které pomáhají tvůrci webových stránek a aplikací poměrně jednoduše a efektivně vytvořit působivé uživatelské rozhraní, které je pěkné a zároveň funkční.

Jeho použití je velmi vhodné v situaci, kdy vývojář či vývojářský tým nemají grafické dovednosti či finanční prostředky na vytvoření prvků uživatelského rozhraní na míru. Bez použití frameworku jako je Bootsrap by za dané situace výsledek jejich práce nemusel působit na uživatele příliš atraktivně a naneštěstí pro běžné programátory (bez grafických dovedností), dnešní uživatelé vnímají kvalitu webové stránky či systému z velké míry podle designu.

Samozřejmě ani použití Bootstrapu není samospasné. Bootstrap totiž poskytuje pouze nástroje a komponenty, z nichž je potřeba správně sestavit výsledný celek. Pro sestavení kvalitního a uživatelsky přívětivého uživatelského rozhraní je tedy stále potřeba jisté grafické cítění a zkušenost. V dnešní době navíc velice narostlo na důležitosti takzvané User eXperience (UX), což by se dalo popsat jako důraz na uživatelskou přívětivost - aby se systém snadno používal, dobře se v něm orientovalo, atd.

Bootstrap je založen na technologiích HTML a CSS, avšak existuje k němu řada dalších komponent založených na JavaScriptu. Jeho použitím tedy lze získat opravdu komplexní balík zvyšující kvalitu webové stránky či aplikace. Jednotlivé komponenty jsou například ikonky, tlačítka, vysouvací nabídky, formulářová pole, atd. Další možností, kterou Bootstrap poskytuje je responzivita, tedy schopnost přizpůsobit velikost prvku velikosti obrazovky.

4 Vlastní zpracování

V této kapitole bude popsána implementace systému včetně vysvětlení jeho součástí. Nejprve bude popsán návrh a rozložení systému, následně budou detailněji rozebrány moduly a fungování určitých procesů a nakonec bude popsána databáze a postup nasazení.

4.1 Vize a analýza

Systém bude mít poměrně evidenční povahu, protože jeho cílem je uchovávat informace a zpřístupňovat je většímu množství aktérů za uplatnění jistých přístupových práv. V systému budou evidovány informace o společnostech, jejich zaměstnancích a projektech. Pod jednotlivé projekty budou spadat úkoly a chyby („bugy“) a rovněž tickety pro komunikaci uživatelů formou zpráv.

Pro realizaci této formy systému je velice vhodná forma webového systému, který umožňuje velice dobrou přístupnost z široké škály zařízení i míst a k jeho využívání není potřeba žádný speciální program, systém, nebo komponenta.

Aktéry, kteří budou se systémem pracovat, lze rozdělit na 2 základní skupiny podle toho, jakou roli hrají v procesu realizace projektu. Další rozdělení už představuje konkrétní role jednotlivých uživatelů, což se projeví v oprávněních k určitým činnostem. Konkrétní rozdělení je uvedeno níže.

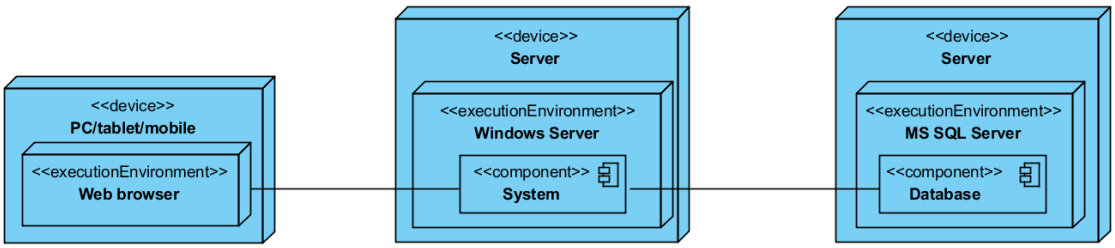
- Strana zpracovatele/dodavatele projektu
  - Admin - administrátor systému
  - Manager - aktér s vyšším oprávněním (např. projektový manažer)
  - Worker - běžný pracovník (např. programátor)
- Klientská strana
  - Manager - aktér s vyšším oprávněním na straně klienta
  - Worker - běžný pracovník na straně klienta

Přístup k systému bude díky internetu umožněn prakticky kdykoliv, což umožní flexibilitu a okamžitou dostupnost dat v případě potřeby. Časový rozměr bude reprezentován stavy jednotlivých evidovaných entit. Smazání používané entity nebude umožněno, protože by to mohlo způsobit nekonzistenci v datech a narušit zpětnou dohledatelnost například v již ukončených projektech.

Díky tomuto systému dokáže dodavatelská firma efektivně spravovat zpracovávané projekty a zároveň komunikovat s osobami na straně klienta.

4.2 Návrh a rozložení systému

Systém je rozdělen do několika celků, které lze nazvat klientská vrstva (client layer), serverová vrstva (server layer) a databázové úložiště.



Obrázek 4: Diagram nasazení

Jednotlivé vrstvy mezi sebou komunikují na principu požadavek-odpověď, kdy požadavky vychází vždy z vyšší vrstvy směrem k vrstvě nižší. Zjednodušeným příkladem je situace, kdy se klientská strana dotazuje serverové strany na data a serverová strana vyžádá data pomocí specifického dotazu z databáze. Data pak putují vždy jako odpověď na zaslaný požadavek neboli dotaz. Neexistuje zde žádná aktivita vyšší vrstvy vyvolaná nižší vrstvou. To je dáno také faktem, že nižší vrstva ani neumí zahájit komunikaci s vrstvou vyšší.

Jako velikou výhodu lze vnímat nezávislost (do jisté míry) jednotlivých vrstev. Určité závislosti mezi použitými technologiemi se vyskytují mezi přímo spolupracujícími vrstvami, tedy klientskou a serverovou a serverovou a databázovým úložištěm. Tyto závislosti se týkají pouze částí zodpovědných za komunikaci se sousední vrstvou, u nichž je jistá závislost logická. Při změně typu databázového úložiště by tedy sice byly potřeba nějaké změny v serverové vrstvě, ale hlavní logika vrstvy může bez problémů zůstat zachována. Změna technologií na klientské vrstvě je velmi jednoduchá, protože napojení na serverovou vrstvu je díky využití technologie AJAX univerzální. Toto umožňuje taktéž vytvoření nového typu klienta, jakým může být například mobilní aplikace. Díky návrhu architektury a použitým technologiím by bylo napojení této mobilní aplikace relativně jednoduché.

Pro ilustraci rozložení systému může sloužit jednoduchý diagram nasazení 4.

4.2.1 Klientská vrstva

Tato část systému běží výhradně na straně klienta a celá logika je tedy zpracovávána v samotném internetovém prohlížeči. Technologie, které zde byly použity, jsou především jazyk TypeScript, který společně s JavaScriptovou knihovnou jQuery zajišťuje veškerou dynamiku. Samozřejmostí je také použití základních webových technologií jako značkovací jazyk HTML5 pro základní rozvržení a Kaskádové styly (CSS) pro vzhled prvků. Pro jednodušší, přívětivější a pěknější uživatelské rozhraní byl použit front-end framework Bootstrap.

Klientská strana je ta část systému, se kterou přichází uživatel do styku a která přímo ovlivňuje jeho dojem. Tato část systému obstarává zobrazování dat v různých formách, podle konkrétní situace. Když pomineme základní rozvržení (layout), jde především o vykreslování tabulek s daty, formulářových prvků, ...

Data, která klientská strana zobrazuje uživateli, jsou získána ze strany serverové. Toto propojení zajišťuje technologie AJAX. Jakmile se uživatel přesune do určité části systému, kde jsou potřeba nějaká data z databáze, je vyslán AJAX požadavek (request) přímo na konkrétní část serverové strany. AJAX požadavek obsahuje určité parametry, podle kterých se přizpůsobí navrácená data. Tento model je výhodnější, než situace, kdy se při každém požadavku na data obnoví celá stránka včetně částí, které ale vždy zůstávají stejné (například menu, základní rozvržení, atd.). Při využití AJAXu se totiž ze serveru nepřenáší všechna data, ale pouze ta, která se mění. Po obdržení jsou nová data dynamicky zapsána do stránky na místo, kam patří. Prakticky se tedy například změní pouze informace v tabulce a vše ostatní zůstane stejné.

4.2.2 Serverová vrstva

Hlavním úkolem serverové strany je obsluhování požadavků z klientské strany, které přichází v podobě AJAX požadavků. Prováděná činnost ve většině případů zahrnuje přijetí parametrů, jejich zpracování, komunikaci s databází, přípravu dat a jejich navrácení zpět klientské straně.

Ústřední technologií serverové strany je ASP.NET a pro objektově-relační mapování (ORM) je použit Entity Framework.

4.2.3 Databázové úložiště

Všechna evidovaná data o entitách v systému jsou uložena v relační databázi, což je pro systém evidenčního charakteru s různě propojenými entitami ideální typ úložiště.

Konkrétním typem databáze je MS SQL Server. Bližší popis návrhu databázové vrstvy je uveden v další části práce 4.5.

4.3 Moduly

V této části jsou popsány součásti systému, které vždy poskytují určitou funkcionalitu, která je pro systém jako celek velmi důležitá. Tyto součásti jsou v kódu reprezentovány třídami, eventuálně soustavou spolupracujících tříd.

4.3.1 Komunikace se serverem - třída DataSource a její specifičtí potomci

Komunikaci se serverem za účelem získání či zaslání dat je vhodné umístit na jedno místo. V tomto systému k tomuto účelu slouží třída DataSource a její potomci. Tito potomci jsou vždy specifičtí pro danou entitu (projekt, uživatel, ...).

Metody těchto tříd obsahují kód související s technologií AJAX, která je použita pro asynchronní komunikaci se serverovou částí systému. AJAX požadavky musí být zasílány na přesnou adresu a rovněž musí být správně nastaveny parametry, které jsou součástí tohoto požadavku a které serverové části upřesňují požadovanou činnost.

V rodičovské třídě DataSource jsou kromě referencí na spolupracující instance nadefinovány také atributy představující nastavení datového zdroje. Toto nastavení (ve formě

souhrnu parametrů) je zasíláno spolu s každým požadavkem na vrácení souhrnu záznamů.

Zmíněné nastavení představují konkrétně položky: *orderBy* (atribut, podle kterého data řadit), *orderHow* (sestupně/vzestupně), *search* (textový řetězec, který uživatel zadal do vyhledávání), *status* a další položky podle použitých filtrů. Pro účely stránkování zde jsou rovněž hodnoty *from* (od kolikátého záznamu vybírat data) a *count* (kolik záznamů vrátit).

4.3.2 Tabulkové přehledy dat - třída TableView

V systémech evidenční povahy je velmi časté vypisování údajů o evidovaných entitách v podobě tabulek, protože to poskytuje značnou přehlednost. O vykreslování seznamů entit v tabulkové podobě se stará třída TableView. Tato třída vznikla zejména z důvodu univerzálního použití pro přehledy všech entit, což bude dále vysvětleno na principu fungování.

Je-li potřeba vykreslit tabulku s přehledem entit (například seznam projektů), je zkonstruována instance třídy TableView a dalších potřebných tříd (budou probrány dále). V konstruktoru je předáno pole, jež nese informace o sloupcích, které budou vypisovány (název sloupce a označení jeho dat v souhrnu dat, která jsou obdržena ze serveru).

Tato instance je podle konkrétní entity (zde projekt) napojena na správný zdroj dat, kterým je specifická instance potomka třídy DataSource (v tomto případě DataSourceProject). Na něm zavolá metodu GetData(). Po přijetí dat (asynchronní operace) se data zpracují (z formátu JSON) a započne vykreslení tabulky. Tabulka je vykreslována postupně po řádcích a vždy jednotlivé sloupce jsou konstruovány podle pole, které bylo předáno v konstruktoru.

Třída TableView umožňuje ve spolupráci s napojeným zdrojem dat také řazení položek. V konstruktoru je totiž kromě pole s informacemi o sloupcích předáno také pole s informací, podle kterých sloupců lze řadit. Hlavičky těchto vyjmenovaných sloupců jsou poté vykresleny jako klikací a po kliknutí se iniciuje řazení. Po dalším kliknutí se vždy přepne pořadí řazení dle daného atributu mezi sestupným a vzestupným.

Univerzálnost použití této třídy spočívá v tom, že ji lze napojit na jakéhokoliv potomka třídy DataSource a tudíž bez jakékoliv změny lze při správném napojení vypisovat tabulkový seznam jakéhokoliv druhu entity. Taktéž vykreslené sloupce lze velice snadno změnit beze změn v samotné třídě, protože stačí upravit pole s informacemi o sloupcích, které je třídě TableView předáno a zajistit, že ze serveru dorazí příslušná data a třída TableView s vykreslením nebude mít žádný problém. V neposlední řadě lze i vybrané sloupce, podle kterých lze záznamy řadit, změnit pouze změnou v poli předávaném v konstruktoru a v kódu na serveru (aby se požadavek na řazení správně přenesl do databáze).

4.3.3 Seznam zpráv v ticketech - třída MessagesView

Pro výpis zpráv v ticketech se tabulkové uspořádání nehodí a tak bylo nutné vytvořit třídu MessagesView. Ta se od třídy TableView až tak příliš neliší. Ke změnám došlo prak-

ticky pouze v metodě Render, která se stará o vykreslení dat. Zprávy se vykreslují jedna po druhé tak, že jsou vždy data konkrétní zprávy (autor, předmět, text, čas, atd.) předány metodě, která je umístí do nadefinované struktury v jazyce HTML. Celá tato struktura včetně dosazených dat je poté vypsána jako jeden záznam do seznamu.

Napojení na potomka třídy DataSource je shodné s třídou TableView. To samé platí pro spolupráci s třídou Paginator, která se i zde stará o stránkování.

4.3.4 Stránkování - třída Paginator

Společně s nutností vykreslení seznamu dat vznikla potřeba na jejich účinné stránkování. Stránkování zvýší přehlednost v záznamech, což jistě uživatelé systému ocení. Navíc má ale stránkování ještě další velmi důležitou roli, která je možná ještě důležitější než předchozí zmíněná. Důležitým přínosem stránkování je totiž redukce přenášených dat. Ze serveru (resp. z databáze) jsou tedy vyžadována a zasílána pouze data, která jsou aktuálně potřeba.

Funkcionalitu stránkování obstarává třída Paginator. Její instance je přímo napojena na instanci třídy TableView, pro kterou stránkování zajišťuje.

Instance třídy Paginator si nejprve vyžádá počet záznamů podle aktuálního nastavení zdroje dat, na který je instance napojena. Pod aktuálním nastavením rozumíme použití různých filtrů a vyhledávání, která mají vliv na počet záznamů a musí být tedy zohledněny i při stránkování. Podle počtu záznamů se pak vykreslí ovládací panel pro stránkování.

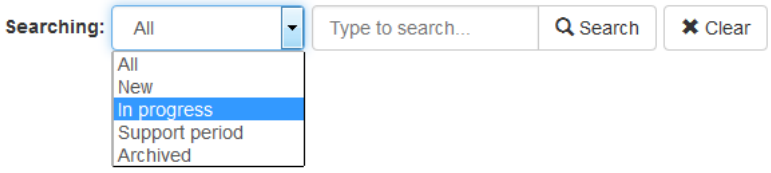
Jakmile uživatel iniciuje změnu stránky, Paginator nastaví zdroji dat hodnoty odpovídající dané stránce - jde o hodnoty *from* (od kolikátého záznamu vybírat) a *count* (kolik záznamů vrátit). Asociovaná instance TableView je poté informována, že došlo ke změně a může si požádat zdroj dat (který už má nyní správné nastavení zohledňující požadovanou stránku) o data. Po obdržení dat se tabulka překreslí a Paginator upraví své ovládací prvky, aby vše odpovídalo aktuální stránce.

Přejde-li uživatel do detailu entity (kterou může libovolně editovat) a vrátí se zpět pomocí navigačních tlačítek, dostane se na původní stránku, na které seznam entit opustil.

4.3.5 Vyhledávání - třída Search

Ve větším množství dat se nelze orientovat bez vyhledávání. K tomu v systému slouží třída Search, která je, stejně jako třída Paginator, napojena na instanci třídy TableView a instanci potomka třídy DataSource. Komponenta tedy zajišťuje vyhledávání v přehledu entit určitého entitního typu (například v souhrnu všech projektů).

Uživatel má možnost vyhledávat pomocí zadání textového řetězce, což slouží především k vyhledávání v názvech či jménech jednotlivých entit. Vyhledávání je za účelem poskytnutí relevantních dat prováděno tak, že se hledá shoda mezi zadaným řetězcem a názvy v databázi, kdy ale název musí daným hledaným řetězcem začínat. Je samozřejmě možné a poměrně jednoduché změnit tento způsob vyhledávání shody na jaký-



Obrázek 5: Vykreslené ovládací prvky komponenty Search

koliv výskyt hledaného řetězce (tedy ne od počátku názvu), avšak při vývoji systému byl upřednostněn prvně zmíněný způsob.

Jelikož téměř všechny entitní typy v databázi obsahují nějakou informaci o stavu, úrovni, typu, atd. je možné jednotlivé entity rozčlenit do skupin právě podle těchto atributů. V praxi tedy jde například o rozdělení ticketů podle stavu na aktivní a dokončené (zjednodušeno). Existuje-li v množině entit takovéto rozdělení, je zde předpoklad, že by v jistých případech uživatel systému ocenil možnost jednotlivé entity/záznamy filtrovat. Z tohoto důvodu byly do komponenty Search zavedeny rovněž filtry. Konkrétní filtry a jejich dostupné hodnoty se odvíjí od konkrétního entitního typu, protože ne všechny mají společné atributy, ve většině případů se tyto atributy liší dostupnými hodnotami, atp.

Jakmile uživatel nastaví řetězec a/nebo filtry a spustí vyhledávání, je toto nastavení, tedy hodnota vyhledávaného řetězce a hodnoty filtrů, přeneseno na datový zdroj a instance TableView je informována, že došlo ke změně a má si vyžádat nová data. Stejně tak se této změně přizpůsobí i přidružená instance třídy Paginator, protože filtrace jistě změní počet dostupných dat a stránkovací komponenta musí toto vzít v potaz. Jakmile je datový zdroj požádán o nová data, odešle AJAX požadavek na serverovou vrstvu. Ve vyslaném požadavku je již promítnuto aktuální nastavení odpovídající vyhledávanému řetězci a hodnotě filtrů. Nastavení datového zdroje lze pak jednoduše změnit změnou zadaného řetězce nebo hodnot filtrů a spuštěním vyhledávání příslušným tlačítkem.

4.3.6 Profil entity - třída DetailItem a její specifičtí potomci

V seznamu záznamů, o jehož vypsání se stará komponenta TableView, je obsažen odkaz na přechod do detailu konkrétního záznamu. Od detailu záznamu se běžně očekává přehled všech údajů, které se k danému záznamu vztahují včetně možnosti editace. Přesně tuto funkcionalitu poskytuje třída DetailItem a její potomci. Rodičovská třída obsahuje společné atributy a chování, ale určité části je třeba přizpůsobit specificky podle konkrétní entity, což je důvod, proč má každá entita svou vlastní třídu, která zmíněnou funkcionalitu poskytuje. V následujícím textu bude používáno označení třídy DetailItem jako označení specifického potomka této třídy.

Třída DetailItem může pracovat ve 3 různých režimech. Tyto režimy jsou obsaženy v enumerátoru (výčtovém typu) ViewMode a jsou pojmenovány *View*, *New* a *Edit*. Použití enumerátoru velmi usnadní přehlednost kódu, protože testování aktuálního režimu

je v kódu poměrně časté. Režim View představuje pouze základní zobrazení, kde jsou všechny údaje vypsány pouze formou textu. Obsahuje-li entita nějakou referenci na jinou entitu (například odkaz na společnost v detailu uživatele), může být například název související entity vykreslen jako odkaz na její detail.

Režimy New (pro vytváření) a Edit (pro úpravu) jsou značně odlišné od prvního zmíněného. Zde již nestačí pouze textové výpisy, ale je nutné vykreslit různé typy vstupních formulářových prvků. Jednotlivé typy se liší podle konkrétního atributu. Existují zde následné typy vstupních polí:

- Textové pole - vstupem je text, přičemž je možné definovat pro vstupní řetězec určitá omezení (např. vyžadování formátu e-mailu, ...)
- Textarea - jde o víceřádkový vstup, který je využíván pro delší texty jako popisy, zprávy, atd. U tohoto typu vstupu je nutné ošetřovat správné uložení a zobrazení konců řádků.
- Datum - u tohoto vstupu je využito komponenty Datepicker, která je součástí Bootstrap frameworku a která zobrazí uživateli kalendář, ve kterém stačí vybrat požadovaný den. Díky tomu je vložení data uživatelsky přívětivé, jednoduché a navíc je vždy po vybrání vloženo v nastaveném formátu.
- Dropdownlist - tento typ vstupu je nejčastěji používán pro atributy, které mají omezený počet hodnot, přičemž tyto hodnoty jsou přesně specifikovány. Příkladem můžou být status, typ, atd.
- Picker - jde o komponentu, která byla vytvořena na míru pro tento systém. Její popis je uveden níže.

4.3.7 Výběr entit - třída Picker

V případě, kdy mezi jednotlivými entitami mají existovat vazby, je nutné vyvinout způsob jejich zadávání uživatelem. Právě pro tento účel byla vytvořena komponenta Picker. Jádrem fungování je JavaScriptová komponenta FancyBox a správné propojení existujících komponent.

Picker funguje na principu zobrazení seznamu entit v takzvaném „lightboxu“. Tento seznam je vypsán s pomocí komponenty TableView a je doplněn o ovládací prvky komponent Search a Paginator, takže i v této „vysunuté“ nabídce může uživatel využívat všechny výhody vyhledávání i stránkování, což přináší vysoký uživatelský komfort.

Komponenta umožňuje dva typy výběrů nazvané *SinglePicking* a *MultiplePicking*. U prvního zmíněného platí, že je vybírána vždy jedna entita. Pokud je vybrána další, původní volba se přepíše. Tento typ je využit například při přiřazení společnosti k projektu. MultiplePicking umožňuje vybrat několik entit ve stejné roli. Druhý typ je využíván pro přiřazení uživatelů k ticketům nebo přiřazení společností uživateli, tedy u situací, kde se vyskytují vazby M:N.

Poté, co uživatel vybere konkrétní entitu, je podle nastavených pravidel zavolána správná metoda, FancyBox se uzavře a vybraná entita je uložena. Uložení se liší dle typu

výběru (popsány v předchozím odstavci) a režimu třídy DetailItem. Jde-li o výběr více entit ve stejné roli, je při editaci (mód Edit) nová volba ihned odeslána formou AJAX požadavku na server. Při vytváření nové entity se volby ukládají do pole, které je odesláno na server až společně s daty nově vytvářené entity. V základní situaci, kde v jedné roli figuruje pouze jedna referencovaná entita, probíhá ukládání až při stisknutí tlačítka pro uložení a není zde vlastně žádný rozdíl mezi módy editace (mód Edit) a vytváření nové entity (mód New). V případě módu zobrazení (View) může být vypsán odkaz na detail asociované entity s jejím názvem.

4.3.8 Vlastní atributy - třída CustomAttributes

Ke každé entitě jsou vedeny určité informace. Ty jsou většinou omezeny již při návrhu systému, kdy navrhující osoba posoudí potřebu, relevantnost a vhodnost jednotlivých atributů entity. Pevná specifikace atributů se následně promítne do návrhu databáze a vstupních formulářových polí.

Toto je poměrně běžný model, který ovšem nebere v potaz situaci, kdy by uživatel rád k entitě zadal další informace. To lze jistě vyřešit přidáním atributu „Poznámky“, kam bude možno zadat víceřádkový textový vstup. Takto má uživatel možnost kdykoliv k entitě připsat libovolné informace. Problém je zde však ten, že informace nejsou strukturovány, což se v případě většího množství textu může ukázat jako poměrně velký problém.

Po uvážení výše uvedeného vznikla komponenta nazvaná *CustomAttributes*. Jejím cílem je poskytnout uživatelům možnost přidávat k entitám libovolné informace ve strukturované podobě. Uživatel si tak může k entitě vytvořit a naplnit atributy s vlastním pojmenováním i obsahem.

Zadávání vlastních atributů s hodnotami probíhá v rámci komponenty DetailItem (4.3.6) v módech pro editaci (Edit) a vytvoření nové entity (New). Kontrolní prvek komponenty je zastoupen tlačítkem *Add new attribute*, které přidá do tabulky detailu entity nový řádek o dvou buňkách se vstupními poli pro název nového atributu a jeho hodnotu. Vstupní pole pro hodnotu atributu je vždy víceřádkové, protože se předpokládá možnost vložení delšího obsahu. Jednotlivé existující nebo právě vytvořené vlastní atributy lze v detailu entity (v módech Edit a New) jednoduše smazat pomocí příslušného tlačítka, které je vykresleno v rámci řádku každého atributu.

V detailu entity v módu pro zobrazení (View) pak uživatel rozdíl mezi původními atributy (dané již v návrhu) a vlastními atributy v podstatě nepozná. Jediná odlišnost je, že jsou seskupeny na konci tabulky, ale k dalšímu odlišení není pravděpodobně žádný zvláštní důvod (pokud by odlišení bylo žádoucí, lze jej nastavit velmi jednoduše). Vlastní atributy jsou zobrazeny pouze uživatelům zpracovatelské firmy (provozující tento systém pro správu projektů).

Tato komponenta přináší uživatelům jistou flexibilitu a možnost přizpůsobení systému bez ohledu na atributy navržené v návrhu. Taktéž v případě potřeby nového atributu není okamžitě nutný zásah programátora za účelem úpravy systému. Ten bude nutný až v případě, že by se došlo k názoru, že nějaký nový atribut se plošně používá u

všech entit daného typu, protože pak už nedává smysl, aby jej uživatelé vždy vytvářeli ručně jako vlastní atribut.

4.3.9 Nahrávání souborů - třída FileUploader

Velmi vhodnou a využívanou funkcionalitou může být možnost nahrávat k určitým entitám soubory. Proto byl v systému implementován modul *FileUploader*. Tento modul musí mít část jak na klientské vrstvě, tak na vrstvě serverové.

Část na klientské vrstvě se stará o vykreslování uživatelského rozhraní, interakci s uživatelem a odeslání souboru na serverovou část. Klientská část má dva různé módy – jeden pouze pro zobrazení již nahraných souborů a druhý editační pro nahrávání a případně i mazání existujících souborů. Při pouhém zobrazení souborů je vždy vidět náhled, jde-li o obrázek, případně jen univerzální ikonu souboru. Taktéž jsou vypsány informace o autorovi, času nahrání a jménu souboru.

V editačním módu je navíc možnost existující soubory smazat. Soubor lze k nahrání na server vybrat pomocí standardního vybírání souboru ve vyskakovacím okně. Rozhraní k vyvolání této akce bylo upraveno do atraktivnější podoby stylovaného tlačítka, aby lépe zapadalo do kontextu uživatelského rozhraní. Rovněž lze soubor vybrat uchopením a upuštěním nad vytyčenou oblastí, tedy takzvanou technikou *Drag and Drop*. Tím, že jsou podporovány oba způsoby je docíleno lepší uživatelské přívětivosti, protože má uživatel na výběr, co mu lépe vyhovuje. Tuto kombinaci obou způsobů lze dnes běžně vídat i u velkých internetových stránek.

Jakmile je soubor vybrán, je okamžitě odeslán na server. Uživatel pak může editovat další položky či provádět další akce, avšak v průběhu nahrávání souboru nemůže entitu uložit. Jakmile je soubor úspěšně nahrán, je zobrazen jeho náhled s možností jej smazat.

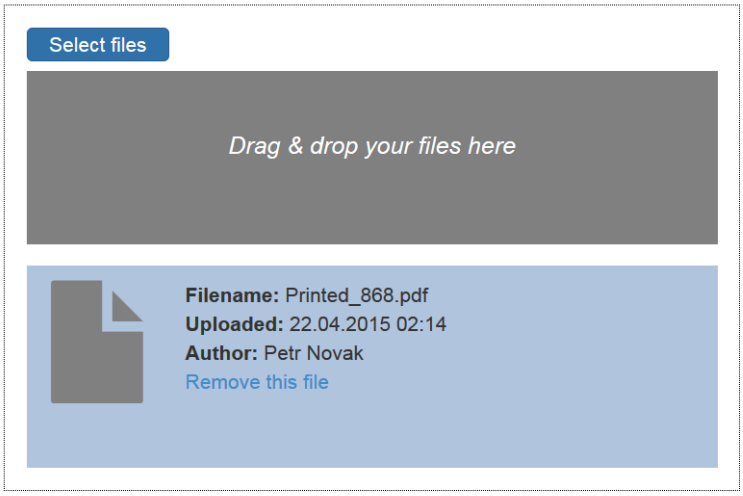
Jakmile serverová část převezme požadavek na nahrání souboru, zjistí, zda již neexistuje soubor stejného jména. Pokud ano, je k názvu souboru přidáno náhodné číslo. Ze souboru „dokument.pdf“ se pak může stát například „dokument_285.pdf“. Po uložení souboru je soubor zapsán do databáze a do klientské části je vráceno ID daného záznamu. Po vytvoření či uložení entity dojde ke spárování souborů s danou entitou.

Aktuálně je možnost nahrávání souborů nasazena u projektů, úkolů a zpráv v ticketech.

4.4 Další součásti a procesy v systému

4.4.1 Komunikace

Důležité komponenty již byly představeny a je tedy možné ilustrovat průběh určité akce v systému. Jako příklad poslouží akce spojené s úkolem. Ilustrace se nachází na obrázku 7 a jde de facto o životní cyklus úkolu. Lze zde vidět vytvoření nového úkolu a následný neomezený počet jeho editací. Jelikož úkol zůstává v systému evidován, není zde znázorněno žádné ukončení životního cyklu a úkol je možné kdykoliv editovat. Dalo by se ale říci, že životní cyklus úkolu končí ve chvíli, kdy již není potřeba k němu přistupovat a nastavením odpovídajícího stavu je archivován.



Obrázek 6: Komponenta FileUploader s náhledem jednoho souboru

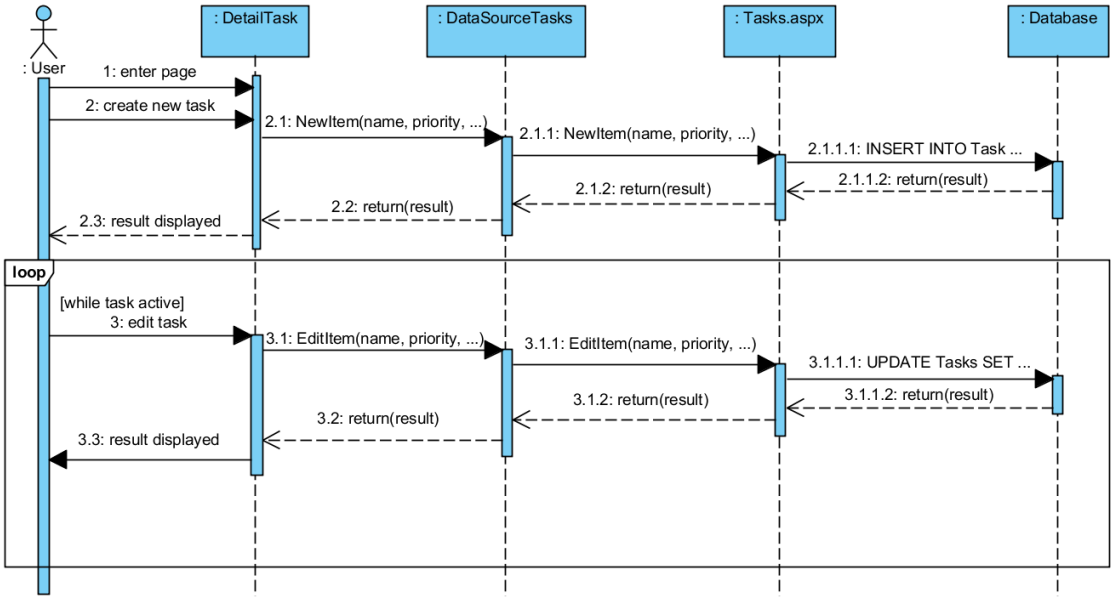
Jak bylo popsáno dříve, komponenta DetailTask (potomek třídy DetailItem) je zodpovědná za interakci s uživatelem ve formě výpisu formulářů, zpráv, přejímání dat, atd. Ke komunikaci se serverem slouží třída DataSourceTasks (potomek třídy DataSource). Stránka *Tasks.aspx* na serveru je zodpovědná za příjem požadavků a jejich zpracování. Vyžaduje-li zpracování požadavku přístup k databázi, je tento přístup zajištěn pomocí ORM nástroje Entity Framework, který vygeneruje SQL dotaz na databázi.

4.4.2 Přihlášení a správa hesel

K přihlášení uživatele do systému je nutné zadání e-mailu a hesla, které musí být v systému evidovány. Jakmile uživatel zadá tyto údaje a iniciuje přihlášení, jsou data poslána v požadavku na server. Tam dojde k ověření, zda v systému existuje uživatel s daným e-mailem. Pokud ano, je ověřena správnost zadaného hesla.

Hesla nejsou v databázi ukládána pouze v textovém tvaru, což by bylo velmi nebezpečné z hlediska bezpečnosti. K jejich zabezpečení je využívána hashovací funkce SHA-256. Tato funkce patří do soustavy hashovacích funkcí SHA-2, jehož předchůdcem je kryptografická hashovací funkce SHA-1, která byla a stále je poměrně rozšířená, avšak nyní je již považována za nedostatečně silnou a tedy málo bezpečnou. Výhodou hashovacích funkcí je to, že i malá změna vstupu (např. změna jednoho písmene) vyústí ve velkou změnu výsledného hashe (zašifrovaného řetězce). Ani u jedné ze zmíněných funkcí nelze z vygenerovaného hashe zpětně získat původní řetězec (v tomto případě heslo).

Existují však postupy, jak díky znalosti hashe původní heslo zjistit. Jedním z těchto postupů může být využití takzvaných „Rainbow tabulek“, které obsahují vypočtené hashe pro různé textové vstupy. Slabé heslo lze pak odhalit poměrně snadno a rychle. Proti tomuto typu prolomení bezpečnosti je v systému nasazena technika používání takzvané „soli“. Tato technika spočívá ve vytvoření řetězce, který vznikne spojením uživatelem



Obrázek 7: Ukázka procesů spojených s úkolem

zadaného hesla a určitého nadefinovaného řetězce. Až tento vzniklý řetězec je pak předán hashovací funkci. I kdybychom tedy zašifrovali tak jednoduchá hesla jako "heslo", "password", nebo "123456", z výsledného hashe nelze původní podobu hesla odhadnout. Sůl lze navíc obohatit o nějaký dynamický prvek specifický pro konkrétní zadané heslo, tudíž může být sůl pro každé heslo unikátní.

Díky uvedeným postupům tedy není nikde v systému ani v databázi uchováváno heslo v podobě, ve které by bylo možné jej přečíst či nějak zpětně zjistit. Navíc ani zašifrovaná podoba hesla nikdy neopouští databázi a serverovou vrstvu, není tedy posílána do klientské vrstvy například spolu s informacemi o uživateli.

Přihlášení je nutné ověřovat nejen při prvotním vstupu do systému, ale i v průběhu využívání systému. Jinak by totiž mohlo být možné vstoupit do systému například díky pouhé znalosti adresy interní stránky. Pokaždé, když je stránka obnovena, je proto vyslán požadavek na server, který ověří, zda je uživatel přihlášen, či nikoliv. Není-li přihlášen, je ze systému přesměrován na přihlašovací formulář. K obnovení stránky může dojít zadáním konkrétní adresy, explicitním vyžádáním (např. klávesou F5 či příslušným tlačítkem ve webovém prohlížeči), nebo například využitím funkce „Zpět“. U uvedeného ověřování přihlášení je ovšem ještě jiné nebezpečí - uživatel by totiž mohl ověřovací požadavek nějakým způsobem vyřadit a systém by neprovedl přesměrování mimo systém.

Tím, co je potřeba v systému chránit jsou data, nikoliv například uživatelské rozhraní. Proto pokud nepřihlášený uživatel nějakým způsobem vstoupí do systému, nevadí, že uvidí, jak systém uvnitř vypadá, ale rozhodně se nesmí dostat k datům. Proto při každém požadavku na data je na serverové straně ověřeno, zda je dotazující se uživatel přihlášen. Pokud ano, jsou mu navrácena požadovaná data (například seznam projektů). Pokud je

zjištěno, že není přihlášen, je zpět do klientské vrstvy poslána pouze informace, která způsobí přesměrování uživatele na přihlašovací formulář. I v případě narušení přesměrování mimo systém však uživatel nezjistí žádná data, což je to nejdůležitější.

4.4.3 Autorizace (oprávnění uživatelů)

Jak je řešeno přihlášení uživatele je popsáno v předchozí části 4.4.2. Při každém požadavku na data je ale rovněž potřeba provést autorizaci. V systému figurují následující úrovně uživatelů:

- Admin - administrátor systému
- Manager - aktér s vyšším oprávněním (např. projektový manažer)
- Worker - běžný pracovník (např. programátor)
- Client manager - aktér s vyšším oprávněním na straně klienta
- Client worker - běžný pracovník na straně klienta

Ne každý typ uživatele smí přistupovat ke všem datům. Jistě bychom nechtěli, aby například běžný pracovník na straně klienta směl přistupovat k údajům o projektu zpracovávaném pro jinou firmu. Pro lepší práci je v některých částech systémů rozdělen tento seznam uživatelských oprávnění do tří kategorií: *SuperRightsUsers* (nejvyšší oprávnění, zahrnuje úrovně Admin a Manager), *HighRightsUsers* (k položkám v rámci kategorie SuperRightsUsers navíc přidává úroveň Client manager), *LowRightsUsers* (nejnižší úroveň, patří sem Worker a Client worker) a *ClientUsers* (zahnuje uživatele ze strany klientské společnosti).

Implementace autorizace funguje tak, že při každém požadavku, kde je její vykonání potřeba, je zjištěna identita uživatele a jeho úroveň. Nejprve bude popsán případ operace, kdy se uživatel nedotazuje na data, ale chce provést nějakou operaci (například editaci entity, vytvoření nové entity, ...). V tomto případě je zjištěna jeho úroveň, a pokud není dostačující, provádění operace je ukončeno zasláním chybové hlášky v rámci odpovědi.

Druhý případ je ten, že se uživatel dotazuje na data. Má-li dostatečná oprávnění na přístup ke všem datům, jsou mu data zaslána. Nemá-li vůbec přístup k daným datům, nejsou mu zaslána žádná data. Pak je zde ještě poslední situace, kdy má uživatel na základě jeho úrovně přístup jen k některým záznamům z množiny vyžadovaného typu dat. V takovém případě jsou v rámci databázového dotazu aplikována příslušná omezení, která zajistí, že uživatel dostane jen jemu přístupná data. Příkladem posledně zmíněné situace může být dotaz na seznam projektů. V případě administrátora nejsou žádná omezení aplikována a je vrácen kompletní seznam všech projektů. V případě běžného pracovníka jsou vráceny pouze projekty jeho společnosti.

Tato forma implementace autorizace umožní, že není nutné dělat velké rozdíly v klientské vrstvě, vše se obstará v serverové vrstvě a klientská vrstva pak jen zobrazí obdržená data. Toto se ale týká především dat (dotazů na data), protože i v klientské vrstvě existují jisté postupy, které se odvíjí od uživatelského oprávnění. Jde ale zejména

o úpravy navigačních prvků uživatelského rozhraní (jejich skrytí, nebo zobrazení) a znemožnění přístupu do určitých sekcí systému. I pokud by se ale uživatel do příslušných sekcí nějak dostal, stejně ze serveru obdrží buď částečná (jemu úmyslně přístupná), nebo žádná data.

4.4.4 Validace vstupních dat

Validace vstupních dat, tedy ověření, zda tato data splňují nastavená kritéria, je velmi důležitá z hlediska bezpečnosti a konzistence. Existují dvě základní umístění validačních procesů – na straně klienta a na straně serveru.

Validace na straně klienta (v prohlížeči) je velmi výhodná v tom, že případné chyby jsou odhaleny okamžitě bez nutnosti odesílat na server jakákoliv data. Tím pádem se šetří výkon serveru i čas uživatele (nemusí čekat na provedení komunikace se serverem). Problém u tohoto typu validace je ten, že nemusí vždy fungovat anebo ji lze jistými způsoby obejít. To znamená, že pokud by byla jediná implementovaná validace vstupních dat na klientské straně, systém by byl zranitelný. Správně by tedy měla být validace dat implementována na obou místech.

Pro validaci na klientské straně je využita jQuery knihovna JQuery Validation Plugin [18]. Tento plugin slouží k poměrně jednoduchému a efektivnímu nasazení validace pomocí JavaScriptu do nově budované i již existující aplikace. Plugin umožňuje poměrně široké možnosti přizpůsobení, protože kromě toho, že obsahuje sadu nadefinovaných validačních metod (např. pro e-mail, URL, atd.), tak také samozřejmě umožňuje napsat si vlastní metody dle potřeby v konkrétním kontextu, což je velice důležité. Chybové hlášky jsou již předem nadefinované v angličtině a přeložené i do dalších 37 jazyků, ale programátor má možnost napsat si vlastní.

Validace vstupních dat v rámci serverové vrstvy probíhá v jednotlivých metodách, které přijímají požadavky z klientské vrstvy. Většina vstupních dat je ověřena pomocí regulárních výrazů, které jsou centrálně nadefinované spolu s metodami, které porovnání výrazu a zadaného řetězce provedou. Také je využíváno metody *TryParse* tříd *int* a *DateTime*. V případě delších textů, kde chceme ponechat uživateli možnost zadávat různé znaky je využito pouze takzvané „escapování“, které převede některé „potenciálně nebezpečné“ znaky na jejich kódové varianty. V případech, kdy je množina přípustných znaků nějakým způsobem omezena, a jsou povoleny jen určité znaky, je uživateli při chybě vypsána rovněž informace o těchto povolených znacích. Chybová hláška s touto informací je nadefinována centrálně na jednom místě, aby bylo snadné ji změnit (například v případě změny množiny povolených znaků). Do této hlášky je navíc z místa zjištění nevalidního vstupu vložena informace o názvu položky, ve které se chyba nachází (například příjmení, název projektu, atd.).

4.4.5 Odesílání e-mailů

Při určitých událostech může systém zaslat zainteresovaným uživatelům e-mail. O tuto funkcionalitu se stará třída *Email*. E-mailová zpráva podporuje stanovení struktury v

HTML. Jako kódování bylo zvoleno UTF-8. Předmět a text zprávy se nastaví podle konkrétní hodnoty z výčtového typu *NotificationType* a většinou obsahuje stručně specifikovanou událost, ke které došlo. E-mail lze zaslat jednomu, ale i více příjemcům s využitím skryté kopie, aby nemusel být e-mail zbytečně odesílán několikrát.

K odeslání e-mailu je využit SMTP server služby Gmail [19] společnosti Google Inc. K přihlášení je využito uživatelské jméno a heslo k účtu na této službě. K zašifrování spojení využívá SMTP klient technologii SSL.

4.4.6 Konfigurace

Konfigurační nastavení jsou uloženy ve 2 třídách - jedna pro klientskou vrstvu (jazyk TypeScript) a druhá pro serverovou vrstvu (jazyk C#).

Nastavení v rámci serverové vrstvy se týkají především zabezpečení. Jsou zde nadefinovány úrovně uživatelů a s nimi související metody, které umožňují pohodlně zjistit, do které kategorie uživatelských oprávnění příslušný kód uživatele spadá. Dále jsou zde nadefinovány chybové hlášky zasílané klientské vrstvě v případě, kdy uživatel není přihlášen, nebo nemá dostatečná oprávnění. Tyto zprávy jsou používány na mnoha místech, přičemž změna jejich obsahu je díky centralizaci velice snadná. Některá nastavení jsou uchována také v souboru Web.Config v sekci „appSettings“.

Nastavení na klientské vrstvě rovněž obsahují informace související s uživatelskými úrovněmi, stejně jako na serverové vrstvě. Navíc jsou zde obsaženy číselníky stavů a typů různých entit společně s metodami pro získání celého konkrétního číselníku nebo pro překlad kódu stavu či typu entity na jeho název. Všechna data jednotlivých číselníků jsou nyní uložena v kódu, jelikož se nepředpokládá jejich častá změna a při nešetrné změně některých číselníků by mohla hrozit nekonzistence v datech či systémovém chování. Metody pro vrácení obsahu číselníku jsou však implementovány podle návrhového vzoru Lazy Load, takže je možné kdykoliv změnit úložiště těchto dat. V tom případě by totiž stačilo do příslušné části metody napsat kód pro získání dat z nového úložiště a nic jiného by nebylo nutno upravovat.

4.4.7 Vlastní knihovny

V některých částech systému je vyžadována určitá společná funkcionalita. Pokud by byla implementována na každém místě zvlášť, její pozdější změna by byla obtížná. Proto v systému existují třídy, které mají vlastně roli knihoven programátorem napsaných metod.

V rámci klientské části jde o třídu *Common*. V ní jsou implementovány funkce (metody) pro práci s časem jako převod serializovaného formátu data (v sekundách) na datum, případně datum s časem (v hodinách a minutách). Rovněž jsou zde metody pro ošetření konců řádků v rámci textů z víceřádkových textových vstupů, aby bylo možné je korektně uložit do databáze a poté znovu zobrazit (jako text, nebo v případě editace uvnitř textového vstupu).

Ve třídě „ServerMethods“, která slouží jako knihovna funkcí (metod) na serverové vrstvě jsou nadefinovány regulární výrazy pro jednotlivé typy vstupů (text, e-mail, atd.) a poté metody, které s nimi pracují jako například *ValidateEmail* a *ValidateDate*.

4.5 Databáze

Pro persistentní uložení informací evidovaných v tomto systému je používána relační databáze. Tento typ databázového systému je za dané situace pravděpodobně tím nejlepším řešením.

4.5.1 Relační databáze

Relační databázový systém uchovává data v podobě tabulek skládajících se z řádků a sloupců. Každý řádek zde představuje jeden záznam a obsahuje tedy informace spolu související, které se většinou vztahují k jedné entitě. Sloupce představují atributy s určitým nadefinovaným datovým typem. Datových typů existuje větší množství, aby pokryly základní potřeby, ale mezi různými databázovými systémy se datové typy nezřídka liší. V jednotlivých buňkách tabulky je vždy právě jedna hodnota specifikovaného datového typu, která má nějaký vztah k ostatním hodnotám v rámci stejného řádku (záznamu).

Ve většině případů náleží každému entitnímu typu jedna tabulka. Vztahy neboli vazby mezi tabulkami (entitními typy) jsou reprezentovány pomocí speciálních atributů, které se nazývají „cizí klíče“. Hodnota pole, které je definováno jako cizí klíč, zpravidla odpovídá primárnímu klíči v záznamu jiné tabulky, se kterým je původní (odkazující) záznam v nějakém vztahu.

Mezi relační databázové systémy patří například Microsoft SQL Server, Oracle, MySQL. Pro tento systém byl zvolen databázový systém Microsoft SQL Server.

Dotazovacím jazykem pro relační databáze je SQL, tedy Structured Query Language. Tento jazyk je standardizován jako ANSI standard, nicméně jeho implementace jednotlivými databázovými systémy se v pokročilejších příkazech občas liší. Díky SQL lze zasílat na databázi dotazy, ve kterých přesně specifikujeme, jaká data chceme obdržet. Možnosti SQL sahají mnohem dále, než k pouhému „vytažení“ určitých hodnot, protože data je možné už na straně databáze různě seskupovat, řadit, provádět různé výpočty, apod.

Tento informační systém má evidenční povahu, což znamená, že jeho hlavním cílem je uchovávat data evidovaných entit za současného dodržení reprezentace vazeb mezi jednotlivými entitami. K tomuto cíli se relační databáze výborně hodí. Je ale pravda, že použitím relační databáze nastane situace, kdy máme vedle sebe objektové prostředí (C# na serverové vrstvě) a relační prostředí (databáze). Tyto dvě prostředí mají určité rozdíly a k jejich překonání je nutno použít takzvané ORM (objektově-relační mapování). Popis ORM a konkrétního zvoleného nástroje pro jeho realizaci v tomto systému (Entity Framework) lze nalézt v části 3.7.

4.5.2 Návrh databáze

Pro návrh databáze byl použit program Oracle SQL Developer Data Modeler, jež umožňuje pomocí uživatelsky přívětivého grafického rozhraní vytvořit model databáze, ve kterém jsou nadefinovány jednotlivé tabulky, jejich atributy se specifikovanými datovými typy a také reprezentace vazeb mezi tabulkami. Z takto nadefinovaného modelu lze

následně vygenerovat SQL kód pro vytvoření (i smazání) požadované databáze. Konceptuální model databáze zobrazující všechny tabulky, jejich atributy a vzájemné propojení tabulek je zobrazen na obrázku 8.

4.6 Nasazení

K nasazení systému je nejprve potřeba připravit potřebné zázemí. Tyto požadavky klade především serverová část systému. Její hlavní technologií je ASP.NET, tudíž bude potřeba server či hosting, který tuto technologii podporuje. Jako databázový systém slouží MS SQL Server, který je rovněž potřeba zajistit.

Je-li připraveno vše potřebné, dalším krokem je vytvoření a naplnění databáze. Pro obojí stačí nad databází spustit dotazy uložené v souboru *install.sql*, které se postarají o vytvoření tabulkové struktury v databázi a také naplnění počátečními daty. Poté již stačí získat „Connection string“, což je řetězec, který slouží k připojení k určité databázi, a uložit jej v konfiguračním souboru *Web.config*. V tomto souboru je taktéž nutné zadat údaje pro e-mailový účet - adresu SMTP serveru, port, e-mailovou adresu a heslo. Také zde lze zasílání e-mailových upozornění zablokovat.

K připojení k serveru již nyní není potřeba nic více, než internetový prohlížeč, kam zadáme adresu systému (adresa závisí na konkrétním nasazení). Přihlašovací údaje k výchozímu administrátorskému účtu jsou uvedeny v tabulce 1.

Uživatelské jméno:	admin@localhost.xyz
Heslo:	adminX58

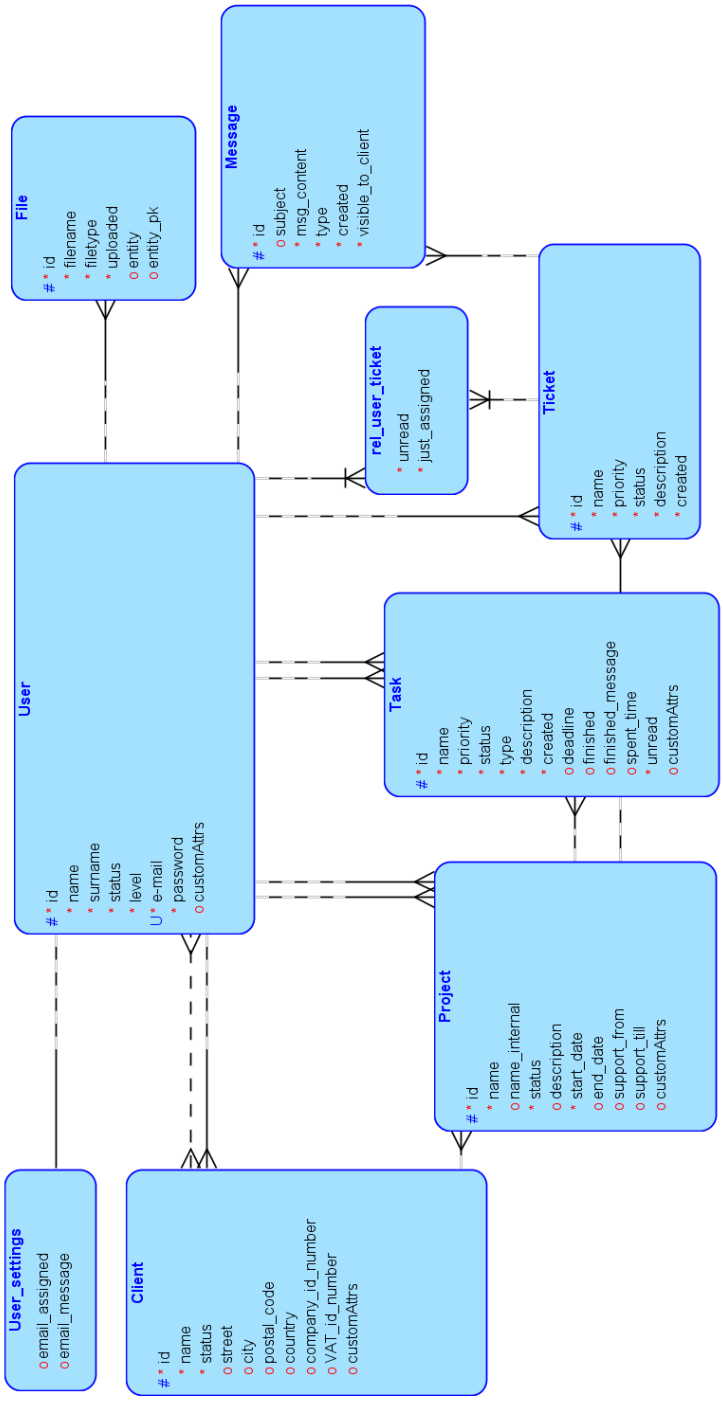
Tabulka 1: Přihlašovací údaje administrátorského účtu

Po prvotním přihlášení je doporučeno provést následující kroky:

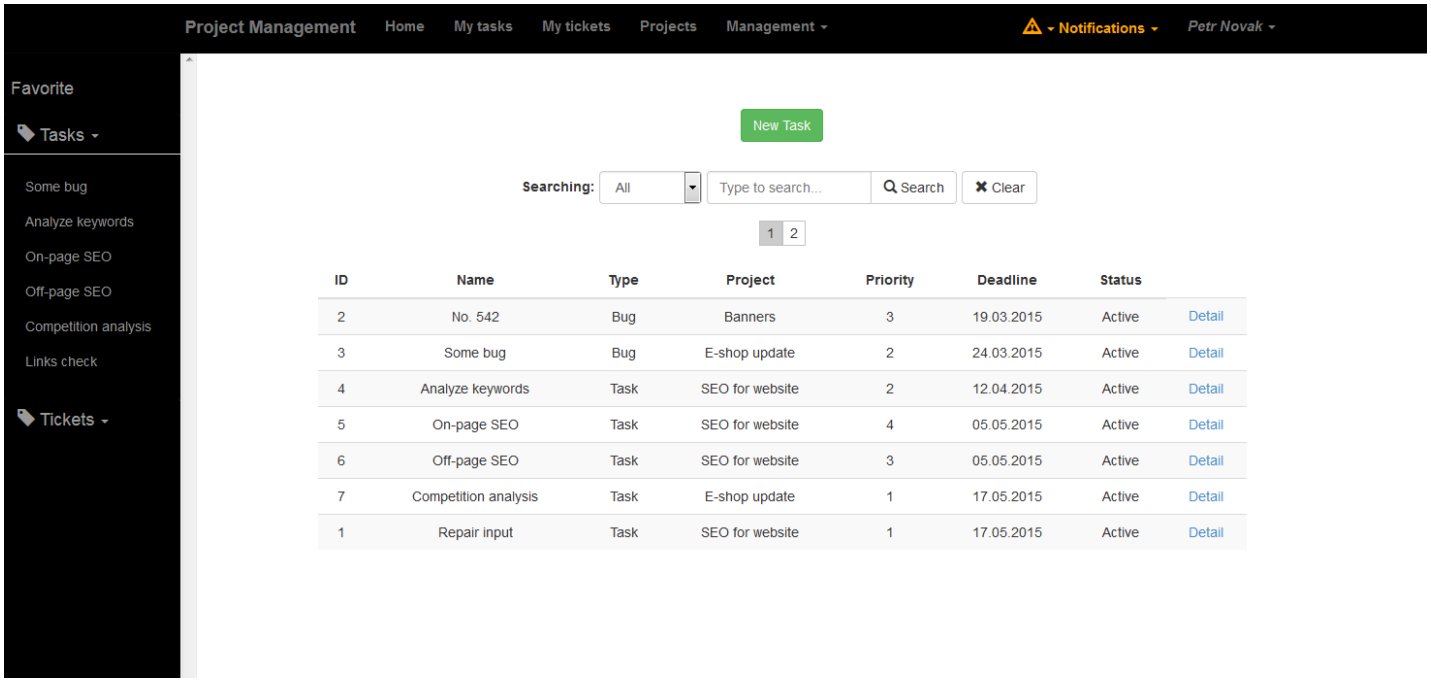
1. Upravit v systému výchozí společnost jejím přejmenováním a zadáním správných informací.
2. Vytvořit další uživatelské účty s oprávněním administrátora pro osoby, které budou vystupovat jako správci systému.
3. Není doporučeno měnit a běžně používat výchozí administrátorský účet. Jedinou doporučenou změnou je změna výchozího hesla. Nové heslo je však nutné velice dobře uschovat.
4. Dále lze vkládat další entity jako uživatele, společnosti, projekty, ...

4.7 Ukázka ze systému

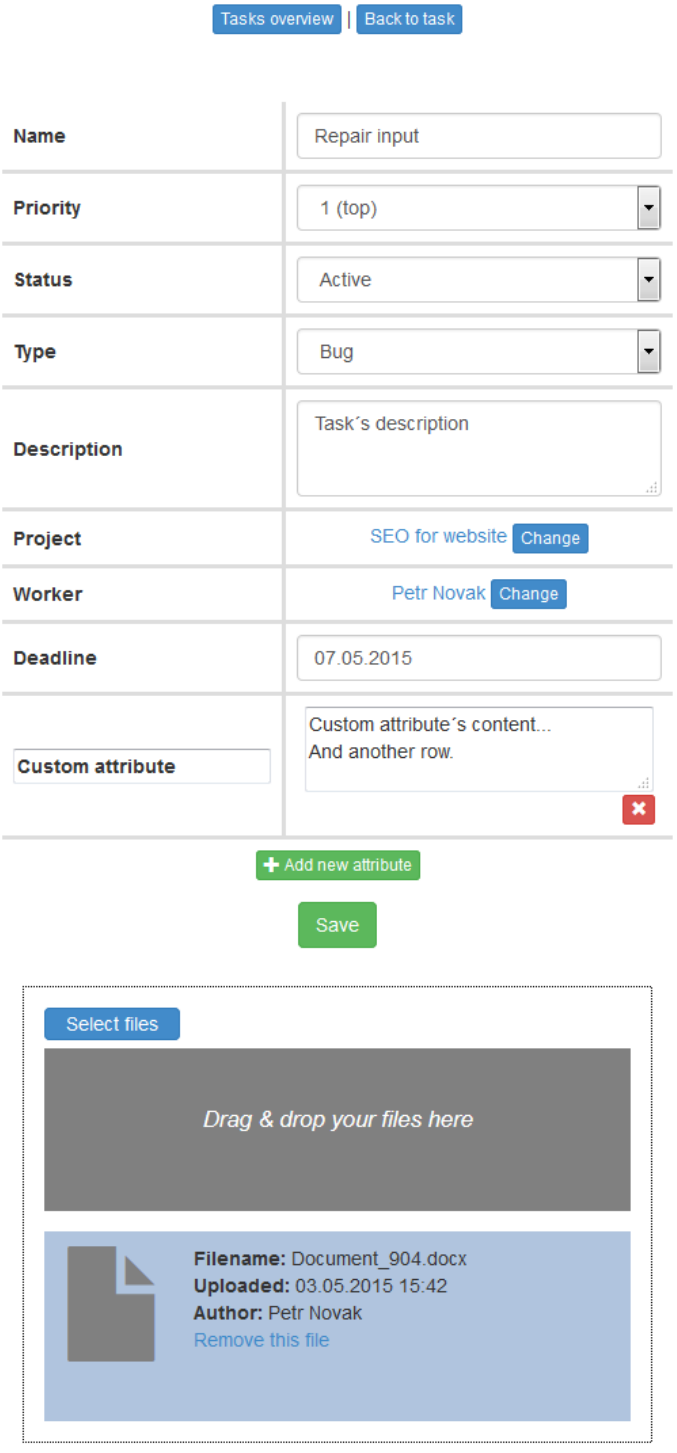
V této kapitole budou předvedeny ukázky ze systému. Na obrázku 9 lze vidět modul TableView společně s ovládacími prvky modulů Search a Paginator. Obrázek 10 ukazuje editaci úkolu a lze na něm vidět moduly DetailView (konkrétně DetailTask), CustomAttributes, FileUploader a rovněž ovládací prvky komponenty Picker.



Obrázek 8: Konceptuální model databáze



Obrázek 9: Ukázka ze systému



Obrázek 10: Ukázka ze systému

5 Závěr

Cílem práce bylo vytvořit systém pro správu projektů, čehož se podařilo dosáhnout. Systém dokáže evidovat klienty, projekty a uživatele (zaměstnance). Ke každému projektu je navíc možné přidat úkoly, chyby, tickety a přílohy. Systém také umožňuje komunikaci s klienty v rámci issue tracking systému.

Zvolené technologie se ukázaly jako vyhovující. Práce s nimi sice občas odhalila jisté problémy, ale ty se dříve či později dařilo řešit. Především bych rád vyjádřil své nadšení z použití jazyka TypeScript a knihovny jQuery oproti „čistému" JavaScriptu. Kombinace obou zmíněných technologií totiž umožňuje pohodlný, efektivní a relativně příjemný vývoj aplikací. Není nutné psát zdlouhavé konstrukce a je možné využít konstrukce nové, které programátoři znají z jiných programovacích jazyků, díky čemuž je pro ně vývoj přirozenější.

Jelikož jsou od sebe klientská a serverová vrstva logicky odděleny, bylo nutné tomu na obou místech vše přizpůsobit. Implementace částí souvisejících s technologií AJAX, která se stará o komunikaci mezi těmito dvěma vrstvami, byla sice poměrně zdlouhavá, ale do budoucna by mohlo jít o velice dobrý krok. V případě, že by to bylo potřeba, lze k systému vytvořit další klientskou část například v podobě mobilní aplikace. Napojení této aplikace na serverovou část by pak nebylo ničím složitým, protože by stačilo z aplikace pomocí AJAXu volat existující metody na serveru. Ty by se vůbec nemusely zabývat tím, zda uživatel používá webový prohlížeč, nebo mobilní aplikaci a fungovaly by stále stejně (samozřejmě pokud by stačilo vracet stejná data).

Velice obohacující byla snaha o centralizaci funkcionalit a univerzálnost napsaných komponent, tedy o to, aby napsaný kód bylo možné využít opakovaně i v lehce odlišných situacích. Tento postup je sice náročnější na promyšlení, návrh a vývoj a taktéž se mohou objevit jisté implementační problémy, ale značná výhodnost se projeví v případě potřeby provést nějaké změny. V tom případě totiž může stačit najít jedno místo, kde provedeme změnu, a změna se pak aplikuje všude, kde je daný prvek použit. V opačném případě by totiž bylo třeba provést změnu na několika místech, což zahrnuje nutnost hledání, několika úprav a otestování, přičemž s rostoucím počtem změn roste i pravděpodobnost výskytů chyb.

Jako možné rozšíření systému lze zmínit například soustavu prvků pro sledování odpracované doby. Aktuálně je možné u úkolu zadat celkovou dobu strávenou nad daným úkolem, což je pouze naprostý základ. Zadané časy by se daly seskupovat do různých souhrnů obohacených o grafy, atd. Tím pádem by vedoucí pracovníci i samotní zaměstnanci mohli mít detailní přehled co a kdy dělali, jak dlouho jim to zabralo, a podobně. Jako další možnost lze ještě zmínit funkci kalendáře, kam by byly zapsány všechny položky s časovou informací (např. deadline úkolu, konec podpory projektu a další).

A Reference

[1] *Basecamp is everyone's favorite project management app.* [online]. [cit. 2015-03-20]. Dostupné z: https://basecamp.com/

[2] *Řízení projektů online | Projektově.CZ* [online]. © 2014 [cit. 2015-03-20]. Dostupné z: http://www.projektove.cz/

[3] *Home :: Bugzilla :: bugzilla.org* [online]. © 1998-2015 [cit. 2015-04-20]. Dostupné z: https://www.bugzilla.org/

[4] *Welcome to TypeScript* [online]. © 2012-2014 [cit. 2015-03-10]. Dostupné z: http://www.typescriptlang.org/

[5] *Home | DefinitelyTyped* [online]. [cit. 2015-03-19]. Dostupné z: http://definitelytyped.org/

[6] ASLESON, Ryan. *AJAX: vytváříme vysoce interaktivní webové aplikace.* Vyd. 1. Překlad Jakub Zemánek. Brno: Computer Press, 2006, 269 s. ISBN 80-251-1285-3.

[7] MICROSOFT CORPORATION. *ASP.NET | The ASP.NET Site* [online]. © 2015 [cit. 2015-04-26]. Dostupné z: http://www.asp.net/

[8] Historical quarterly trends in the usage of JavaScript libraries, April 2015. Q-SUCCESS. *W3Techs - extensive and reliable web technology surveys* [online]. © 2009-2015 [cit. 2015-03-17]. Dostupné z: http://w3techs.com/technologies/history_overview/javascript_library/all/q

[9] THE JQUERY FOUNDATION. *JQuery* [online]. © 2015 [cit. 2015-03-20]. Dostupné z: https://jquery.com/

[10] THE JQUERY FOUNDATION. *JQuery Plugin Registry* [online]. © 2015 [cit. 2015-04-12]. Dostupné z: http://plugins.jquery.com/

[11] *Fancybox - Fancy jQuery lightbox alternative* [online]. [cit. 2015-03-17]. Dostupné z: http://fancybox.net

[12] *Moment.js | Home* [online]. [cit. 2015-03-17]. Dostupné z: http://momentjs.com/

[13] Google Charts — Google Developers. GOOGLE, Inc. *Google Developers* [online]. [cit. 2015-04-22]. Dostupné z: https://developers.google.com/chart/

[14] JSON Tutorial. REFSNES DATA. *HTML Tutorial* [online]. © 1999-2015 [cit. 2015-03-17]. Dostupné z: http://www.w3schools.com/json/

[15] FOWLER, Martin. *Patterns of enterprise application architecture.* Boston: Addison-Wesley, c2003, xxiv, 533 s. Addison-Wesley signature series. ISBN 0321127420.

[16] What is Entity Framework? *Entity Framework Tutorial* [online]. © 2015 [cit. 2015-03-17]. Dostupné z: http://www.entityframeworktutorial.net/what-is-entityframework.aspx

[17] *Bootstrap · The world's most popular mobile-first and responsive front-end framework.* [online]. [cit. 2015-04-14]. Dostupné z: http://getbootstrap.com/

[18] *JQuery Validation Plugin | Form validation with jQuery* [online]. [2014] [cit. 2015-04-12]. Dostupné z: http://jqueryvalidation.org/

[19] GOOGLE, Inc. *Gmail – Bezplatné úložiště a e-mail od Googlu* [online]. [cit. 2015-03-22]. Dostupné z: https://www.gmail.com/intl/cs/mail/help/about.html

[20] MICROSOFT CORPORATION. *TypeScript: Language Specification* [online]. Version 1.4. 2014, October, 2014 [cit. 2015-03-15]. Dostupné z: http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf

[21] FENTON, Steve. *TypeScript for C# Programmers* [online]. First Edition. InfoQ, 2013 [cit. 2015-03-17]. Dostupné z: http://www.infoq.com/minibooks/typescript-c-sharp-programmers

[22] MAHARRY, Daniel. *TypeScript revealed.* Apress, 2013, xix, 83 pages. Expert's voice in .NET. ISBN 978-1430257257.

[23] MARGORÍN, Marián. *JQuery bez předchozích znalostí.* Vyd. 1. Brno: Computer Press, 2011, 253 s. ISBN 978-80-251-3379-8.

[24] MICROSOFT CORPORATION. *MSDN – Microsoft Developer Network* [online]. © 2015 [cit. 2015-04-14]. Dostupné z: https://msdn.microsoft.com

[25] What is TypeScript? Pros and Cons - Designmodo. DESIGNMODO. *Designmodo: Web Design Blog and Shop* [online]. © 2015 [cit. 2015-04-22]. Dostupné z: http://designmodo.com/typescript/

[26] *Relační vs. objektově-relační vs. objektové databáze.* [online]. [cit. 2015-03-10]. Dostupné z: http://www.fi.muni.cz/ xbatko/oracle/compare.html

[27] Práce se soubory v HTML5 - Zdroják. *Zdroják - o tvorbě webových stránek a aplikací* [online]. 22.4.2010 [cit. 2015-04-12]. Dostupné z: http://www.zdrojak.cz/clanky/prace-se-soubory-v-html5/

[28] Čtení souborů javascriptem pomocí FILE API - Root.cz. *Root.cz - informace nejen ze světa Linuxu* [online]. 22. 8. 2012 [cit. 2015-04-12]. Dostupné z: http://www.root.cz/clanky/cteni-souboru-javascriptem-pomoci-file-api/

[29] How to Hash and Salt Passwords in ASP.NET. SECURITY INNOVATION, Inc. *Security Innovation | Software Runs the World - We Secure It! | The Application Security Company* [online]. © 2015 [cit. 2015-05-01]. Dostupné z:

http://web.securityinnovation.com/appsec-weekly/blog/bid/58887/How-to-Hash-and-Salt-Passwords-in-ASP-NET

[30] Simple Sidebar - Bootstrap Sidebar Template - Start Bootstrap. IRON SUMMIT MEDIA STRATEGIES. *Start Bootstrap - Free Bootstrap Themes and Templates* [online]. [cit. 2015-04-18]. Dostupné z: http://startbootstrap.com/template-overviews/simple-sidebar/

[31] Refsnes Data. *HTML Tutorial* [online]. © 1999-2015 [cit. 2015-03-10]. Dostupné z: http://www.w3schools.com

B Disk CD

K práci je přiložen disk CD, který obsahuje elektronickou verzi tohoto dokumentu a zdrojové kódy aplikace.

- Adresář *Text* - obsahuje text bakalářské práce ve formátu PDF.
- Adresář *Aplikace* - obsahuje zdrojové kódy vyvinutého systému.
- Adresář *Databáze* - obsahuje SQL dotazy k vytvoření databáze.